elektor books

Raspberry Pi 5 Essentials

Program, build, and master over 60 projects with Python

TRIED · TESTED

Dogan Ibrahim

elektor
design > share > earn

Raspberry Pi 5 Essentials

Program, build, and master over 60 projects with Python

Dogan Ibrahim

elektor
design > share > earn

This is an Elektor Publication. Elektor is the media brand of
Elektor International Media B.V.
PO Box 11, NL-6114-ZG Susteren, The Netherlands
Phone: +31 46 4389444

Declaration

The author, editor, and publisher have used their best efforts in ensuring the correctness of the information contained in this book. They do not assume, and hereby disclaim, any liability to any party for any loss or damage caused by errors or omissions in this book, whether such errors or omissions result from negligence, accident or any other cause. All the programs given in the book are Copyright of the Author and Elektor International Media. These programs may only be used for educational purposes. Written permission from the Author or Elektor must be obtained before any of these programs can be used for commercial purposes.

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

ISBN 978-3-89576-586-5 Print
ISBN 978-3-89576-587-2 eBook

Editor: Clemens Valens
Prepress Production: D-Vision, Julian van den Berg
Print: Ipskamp Printing, Enschede (NL)

Contents

Preface

The Raspberry Pi 5 is a credit-card-sized computer from Raspberry Pi that can be used in many applications, such as in audio and video media centers, as a desktop computer, in industrial controllers, robotics, and in many domestic and commercial applications. In addition to the many features found in other Raspberry Pi computers, the Raspberry Pi 5 offers Wi-Fi and Bluetooth 5.0 (with BLE support), which makes it highly desirable in remote and Internet-based control and monitoring applications.

The Raspberry Pi 5 is based on a 64-bit quad-core ARM Cortex-A76 processor running at 2.4 GHz. This implies a performance boost of two to three times compared to the Raspberry Pi 4. Raspberry Pi 5 comes with an enhanced graphic performance, using an 800 MHz VideoCore VII graphics chip. Additionally, the Raspberry Pi 5 features the RP1 southbridge chip made by Raspberry Pi. With the help of this RP1 southbridge, Raspberry Pi 5 delivers higher performance and more functionality for peripheral devices. It should now be possible to carry out many real-time operations such as audio digital signal processing, real-time digital control and monitoring, and many other real-time operations using this tiny powerhouse.

This book is about the Raspberry Pi 5 computer and its use in various control and monitoring applications. The book explains in simple terms and with many tested and working example projects how to configure the Raspberry Pi 5 computer, how to use the latest operating system (Bookworm), and how to write application programs using the popular Python programming language.

The book starts with an introduction to the Raspberry Pi 5 computer and covers the important topics of accessing the computer locally and remotely. Use of the console command language as well as accessing and using the desktop GUI have been described with working examples.

The remaining parts of the book cover many Raspberry-Pi-5-based hardware projects using components and devices such as LEDs, buzzers, LCDs, ultrasonic sensors, temperature sensors, Sense HAT, camera modules, etc. Example projects are given using Wi-Fi and Bluetooth modules to send and receive data from smartphones, from the PC, and sending real-time temperature and atmospheric pressure data to the cloud.

All the projects presented in the book have been tested and are working. Complete circuit diagrams and full program listings are given for each project, with detailed descriptions of the operation of each project. The following subheadings are used in every project wherever necessary:

- Project title
- Project description
- Block diagram
- Circuit diagram
- Program listing
- Suggestions for future work

I hope the readers find the book helpful and enjoy reading it, and use a Raspberry Pi 5 in their next new projects.

Prof Dr. Dogan Ibrahim
London

Chapter 1 • The Raspberry Pi 5

1.1 Overview

The Raspberry Pi 5 is the latest credit card size computer from Raspberry Pi. In this chapter, we will look at the specifications of this new computer and compare it with the Raspberry Pi 4.

1.2 The Raspberry Pi 5

Raspberry Pi 4 was released in June 2019. There has been a long wait for a newer model and finally the Raspberry Pi 5 was launched in October 2023.

The Raspberry Pi 5 is claimed to have two or three times the processing power of The Raspberry Pi 4, which is already a very popular single board computer. The Raspberry Pi 5 is currently available in 4 GB and 8 GB memory capacities, but smaller memory devices may appear later. Although the Raspberry Pi 5 is the same size and shape as the Model 4B, it has a number of interesting new features such as PCIe connector, power button, built-in real-time clock and some others that we will investigate in this chapter.

The Raspberry Pi 5 is based on a 2.4 GHz Cortex-A76 ARM processor with a new south-bridge for handling the peripheral interface. A new VideoCore VII GPU is provided with 800 MHz speed. The dual camera interface is another nice feature of the Raspberry Pi 5. The microSD card interface now supports cards that work at much higher speeds.

Table 1.1 shows a comparison of the Raspberry Pi 4 and 5. Notice that both devices have dual 2 × 4kp60 HDMI display interfaces, although Pi 5 supports HDR output. The 2 × 20 pin GPIO interface is the same in both devices. The Raspberry Pi 5 additionally has two camera interfaces, a PCIe bus connector, a UART interface, an RTC clock power connector, and a fan power connector. Wi-Fi and Bluetooth are supported by both devices. The on-board power switch on Pi 5 is a useful addition and was requested by many users. Pi 5 is powered from 5 V/4 A USB-C type power supply, where Pi 4 is powered from a 3 A power supply. Pi 5 is slightly more expensive than Pi 4.

	Raspberry Pi 4	Raspberry Pi 5
SoC	BCM2711 SoC Cortex-A72 CPU at 1.8 GHz	BCM2712 SoC Cortex-A76 CPU at 2.4 GHz
CPU	4 core	4 core
Instruction set	ARMv8-A	ARMv8-2
Display	500 MHz VideoCore Vi GPU	800 MHz VideoCore VII GPU
L2 Cache	1 MB (shared)	2 MB
L3 Cache	None	2 MB (shared)
RAM	1, 2, 4, 8 GB LPDDR4	4, 8 GB LPDDR4X
SD Card	microSD	microSD (high speed SDR104 compatible)
GPIO	2 × 20 pin	2 × 20 pin

USB ports	2× USB2 2× USB3	2× USB2 2× USB3
Networking	Gigabit Ethernet port	Gigabit Ethernet port
Connectors	2-lane MIPI display port 2-lane MIPI CSI camera port 4-pole stereo audio and composite video port	2× MIPI camera 2× 4-lane MIPI camera/display PCIe 2.0 interface UART port RTC clock power port Fan power port
Wi-Fi/Bluetooth	802.11ac, Bluetooth 5/BLE	802.11ac, Bluetooth 5/BLE
Power button	None	Yes
Power	5 V, 3 A USB-C	5 V, 4 A USB-C
Size	85 × 56 mm	85 × 56 mm

Table 1.1 Comparison of Raspberry Pi 5 and Raspberry Pi 4

There are two micro-HDMI based monitor ports on both devices, with both having the same specifications.

The Ethernet port and USB ports are swapped. As a result of this, the Raspberry Pi 4 case is incompatible with the Pi 5 and a new case is required.

The camera and display connectors on the Raspberry Pi 5 are 15-pin and smaller, instead of the original 22-pin connector used on Pi 4. A ribbon cable with 22-pin on one side and 15-pin on the other side is required to connect an existing Raspberry Pi 4 camera to the Raspberry Pi 5. The Raspberry Pi 5 has two connectors, allowing two cameras or DSI displays (or a mix of either) to be connected. The PCIe connector is for fast external PCIe compatible peripherals, such as SSDs.

The new power button on the Raspberry Pi 5 could be very useful. When the device is On, pressing the button brings the shutdown (logout) menu. A safe shutdown will occur with another press of the power button.

Figure 1.1 shows the front view of the Raspberry Pi 5 with the components labelled for reference.

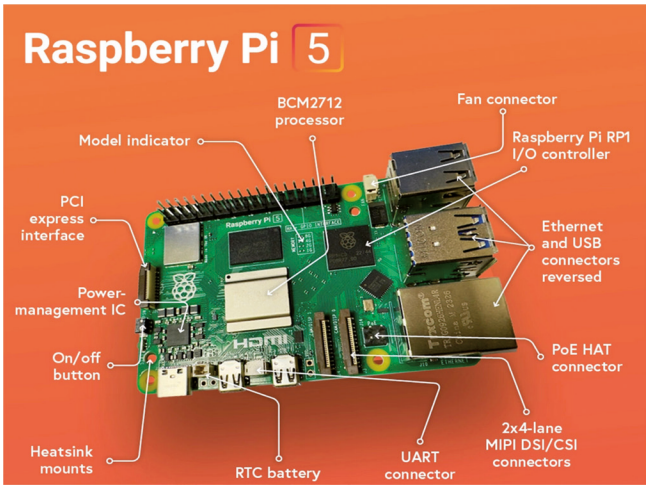

Figure 1.1 Raspberry Pi 5

The Raspberry Pi 5 gets rather hot, and it is recommended to use a cooler to lower the CPU temperature. Although the idle CPU temperature is around 50°C, it can go higher than 85°C under a stress test. An active cooler is available for the Raspberry Pi 5. Holes and power points are provided on the board to install and power the active cooler. Figure 1.2 shows the Raspberry Pi 5 with the active cooler installed. The active cooler cools down the SoC, RAM, and the southbridge chip. When the CPU is idle, the active cooler keeps the CPU temperature at around 40°C. The fan of the cooler operates automatically when the CPU temperature goes just above 50°C.

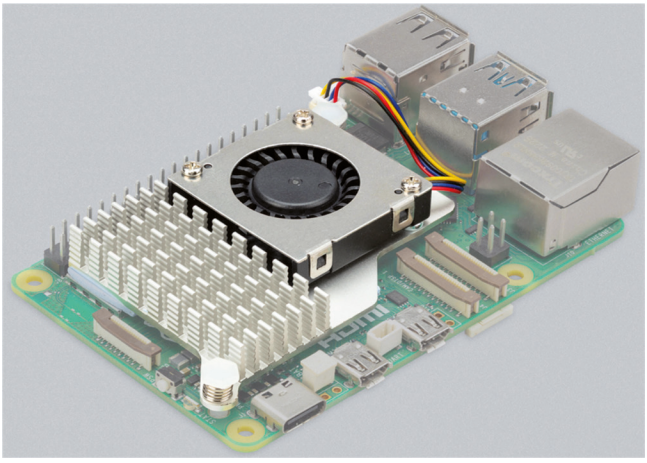

Figure 1.2 The Raspberry Pi 5 with active cooler

The Raspberry Pi 5 operating system (OS) is based upon Debian 12 with the code name **Bookworm**. This OS, released in July 2023, comes with a new Python interpreter (Python 3.11). This means that a Python package cannot be installed using the **pip** commands.

Another major software change is that the RPi.GPIO library (created by Ben Croston) was not available at the time of writing this book. As a result of this, all the GPIO-based Python programs in the book have been developed using the **gpiozero** library. Most third party HATs are based on RPi.GPIO and these will not work until their software is changed by their manufacturers. It is hoped that the manufacturers will change their software by the time Raspberry Pi 5 becomes officially widely available.

Chapter 2 • Installing the Raspberry Pi 5 Operating System

2.1 Overview

The Raspberry Pi 5 operating system **Bookworm** is available either on a pre-installed microSD card, or you can download the operating system image on a blank microSD card. In this chapter, you will learn to install the operating system using both methods.

2.2 Using a pre-installed SD card

The pre-installed Raspberry Pi operating system is available on various sized microSD cards. In this section, the author used the pre-installed 32 GB microSD card supplied by Elektor. Additionally, the author used a 7-inch HDMI compatible monitor, a Raspberry Pi official keyboard, and a mouse. The author's hardware setup between the Raspberry Pi 5 and various devices is shown in Figure 2.1.

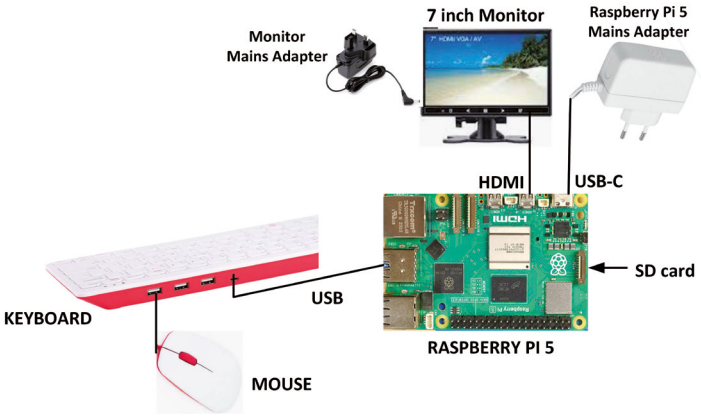

Figure 2.1 The author's hardware setup

The steps are as follows:

- Insert the pre-installed microSD card into your Raspberry Pi 5
- Connect all the devices as in Figure 2.1
- Connect the Raspberry Pi power adapter to the mains supply
- You should see the Raspberry Pi booting the first time and asking you various questions to set up the device, such as the username, password, Wi-Fi network name and password, any updates if necessary, etc. (see Figure 2.2 for some displays on the monitor). In this book, the username is set to **pi**.
- The Raspberry Pi will boot in Desktop mode and will display the default screen. You can press Ctrl+Alt+F1 at any time to change to the Console mode

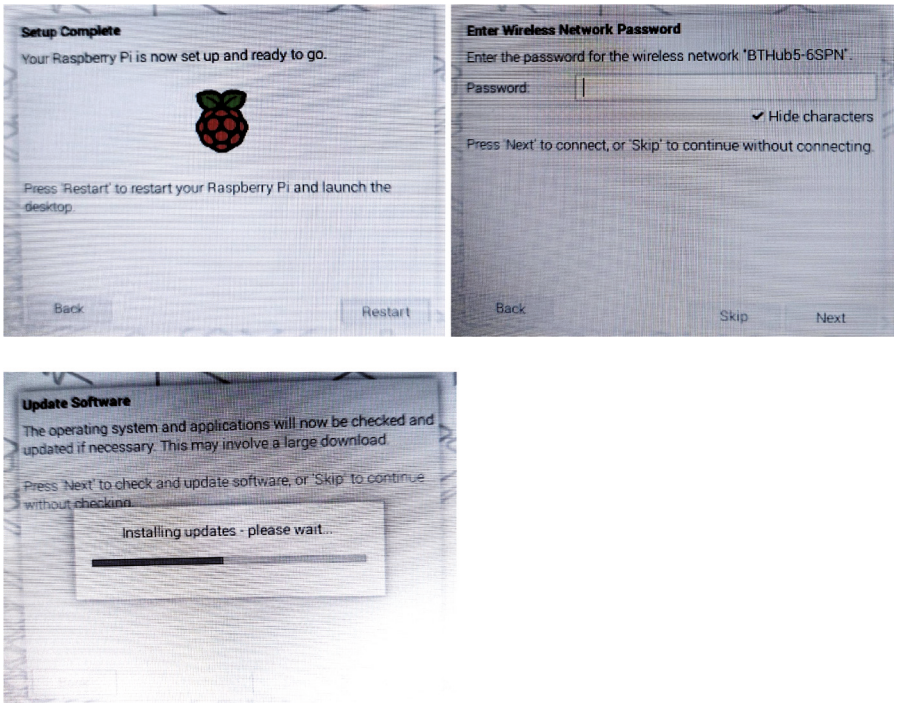

Figure 2.2 Raspberry Pi 5 booting for the first time.

2.3 Larger font in Console mode

It is probably hard to see the characters on a 7-inch monitor in console mode. You can follow the steps below to increase the font size:

- Make sure you are in the Console mode
- Enter the following command:

  pi@raspberrypi: ~ $ **sudo dpkg-reconfigure console-setup**

- Select **UTF-8** in the **Package Configuration** screen (Figure 2.3)

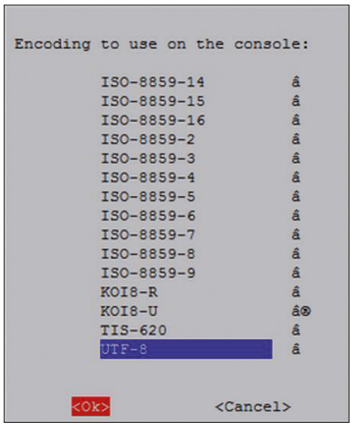

Figure 2.3 Select UTF-8

- Select **Guess optimal character set** (Figure 2.4)

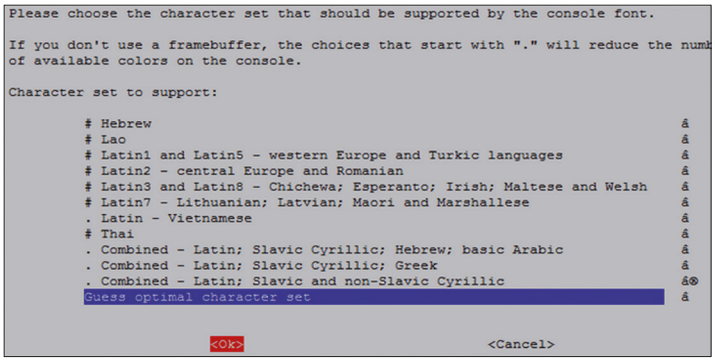

Figure 2.4 Select Guess optimal character set

- Select **Terminus** (Figure 2.5)

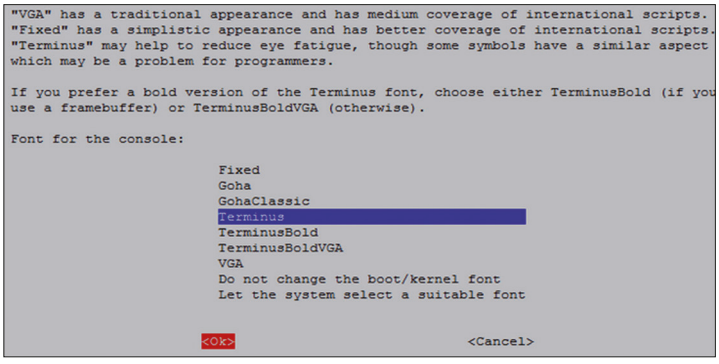

Figure 2.5 Select Terminus

- Select font **16x32** (Figure 2.6)

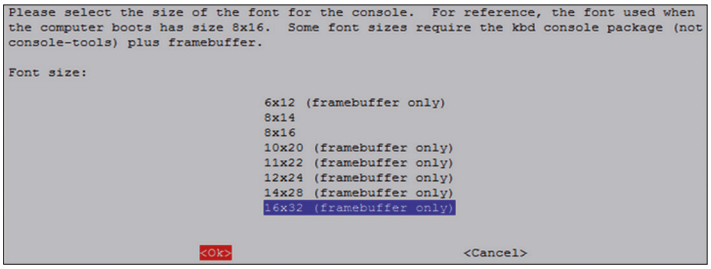

Figure 2.6 Select font 16x32

2.4 Accessing your Raspberry Pi 5 Console from your PC – the Putty program

In many applications, you may want to access your Raspberry Pi 5 from your PC. This requires enabling the SSH on your Raspberry Pi and then using a terminal emulation software on your PC. The steps to enable the SSH are as follows:

- Make sure you are in Console mode
- Type: **sudo raspi-config**
- Move down to **Interface Options**
- Highlight **SSH** and press Enter (Figure 2.7)

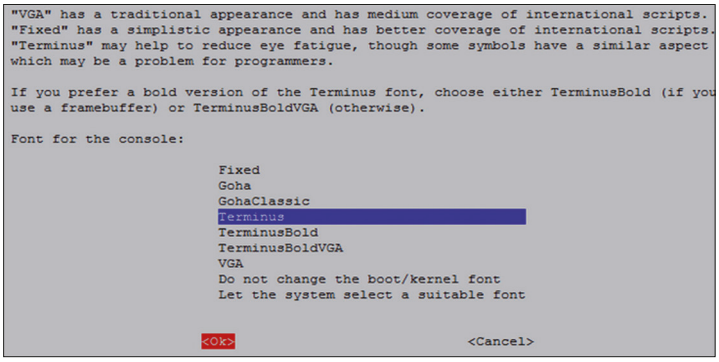

Figure 2.7 Highlight SSH

- Click **Yes** to enable SSH
- Click **OK**
- Move down and click **Finish**

You will now have to install a terminal emulation software on your PC. The one used by the author is the popular Putty. Download Putty from the following website:

https://www.putty.org

- Putty is a standalone program and there is no need to install it. Simply double click to run it. You should see the Putty startup screen as in Figure 2.8.

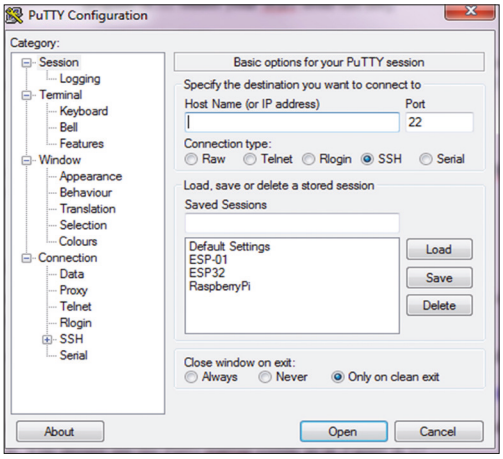

Figure 2.8 Putty startup screen

- Make sure that the Connection type is SSH and enter the IP address of your Raspberry Pi 5. You can obtain the IP address by entering the command **ifconfig** in console mode (Figure 2.9). In this example, the IP address was: **192.168.1.251** (see under **wlan0:**)

```
pi@raspberrypi:~ $ ifconfig
eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether d8:3a:dd:77:b2:e2  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 107

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 105  bytes 9175 (8.9 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 105  bytes 9175 (8.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.251  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 2a00:23c7:868d:7b01:1562:5802:73c0:1ff6  prefixlen 64  scopeid 0x
<global>
        inet6 fe80::966a:a2d8:912:fa6b  prefixlen 64  scopeid 0x20<link>
```

Figure 2.9 Command ifconfig

- Click **Open** in Putty after entering the IP address and selecting **SSH**

- The first time you run Putty, you may get a security message. Click **Yes** to accept this security alert.

- You will then be prompted to enter the Raspberry Pi 5 username and password. You can now enter all Console-based commands through your PC.

- To change your password, enter the following command:

  pi@raspberrypi: ~ $ **passwd**

- To restart the Raspberry Pi, enter the following command:

  pi@raspberrypi: ~ $ **sudo reboot**

- To shut down the Raspberry Pi, enter the following command. Never shutdown by pulling the power cable, as this may result in the corruption or loss of files:

  pi@raspberrypi: ~ $ **sudo shutdown –h now**

2.4.1 Configuring Putty

By default, the **Putty** screen background is black with white characters. The author prefers a white background with black characters, with the character size set to 12 points bold. You should save your settings so that they are available next time you want to use Putty. The steps to configure Putty with these settings are given below:

- Restart Putty
- Select **SSH** and enter the Raspberry Pi IP address
- Click **Colours** under **Window**
- Set the **Default Foreground** and **Default Bold Foreground** colours to black (Red:0, Green:0, Blue:0)
- Set the **Default Background** and **Default Bold Background** to white (Red:255, Green:255, Blue:255)
- Set the **Cursor Text** and **Cursor Colour** to black (Red:0, Green:0, Blue:0)
- Select **Appearance** under **Window** and click **Change** in **Font settings**. Set the font to **Bold 12**.
- Select **Session** and give a name to the session (e.g. MyZero) and click **Save**.
- Click **Open** to open **Putty** session with the saved configuration
- Next time you restart **Putty**, select the saved session and click **Load** followed by **Open** to start a session with the saved configuration

2.5 Accessing the Desktop GUI from your PC

If you are using your Raspberry Pi 5 with local keyboard, mouse, and display, you can skip this section. If, on the other hand, you want to access your Desktop remotely over the network, you will find that SSH services cannot be used. The easiest and simplest way to access your Desktop remotely from a computer is by using the VNC (Virtual Network Connection) client and server. The VNC server runs on your Pi and the VNC client runs on your computer. It is recommended to use the **tightvncserver** on your Raspberry Pi 5. The steps are:

- Enter the following command:

  pi$raspberrypi:~ $ **sudo apt-get install tightvncserver**

- Run the **tightvncserver**:

  pi$raspberrypi:~ $ **tightvncserver**

  You will be prompted to create a password for remotely accessing the Raspberry Pi desktop. You can also set up an optional read-only password. The password should be entered every time you want to access the Desktop. Enter a password and remember your password.

- Start the VNC server after reboot by the following command:

  pi$raspberrypi:~ $ **vncserver :1**

  You can optionally specify screen pixel size and colour depth in bits as follows:

  pi$raspberrypi:~ $ **vncserver :1 –geometry 1920x1080 –depth 24**

- We must now set up a VNC viewer on our laptop (or desktop) PC. There are many VNC clients available, but the recommended one which is compatible with **TightVNC** is **TightVNC** for the PC, which can be downloaded from the following link:

  https://www.tightvnc.com/download.php

- Download and install the **TightVNC** software for your PC. You will have to choose a password during the installation.

- Start the **TightVNC Viewer** on your PC and enter the Raspberry Pi IP address followed by ':1'. Click **Connect** to connect to your Raspberry Pi (Figure 2.10)

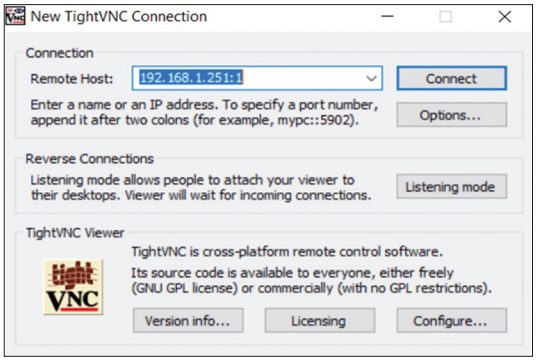

Figure 2.10 Connect to TightVNC Viewer

- Enter the password you have chosen earlier. You should now see the Raspberry Pi 5 Desktop displayed on your PC screen (Figure 2.11)

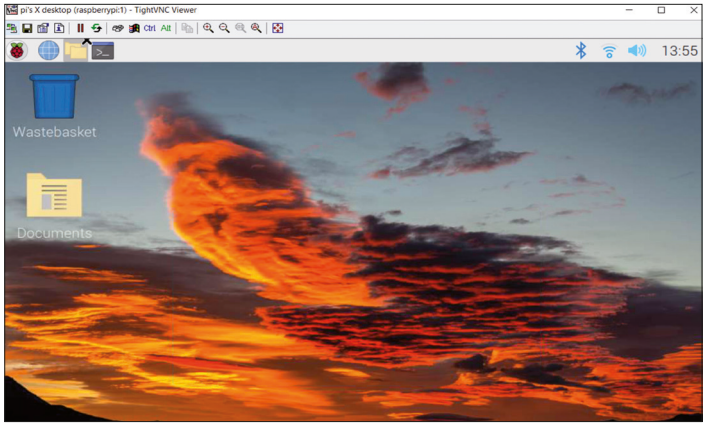

Figure 2.11 Raspberry Pi 5 Desktop

- The VNC server is now running on your Raspberry Pi 5 and you have access to the Desktop GUI.

2.6 Assigning a static IP address to your Raspberry Pi 5

When you try to access your Raspberry Pi 5 remotely over your local network, it is possible that the IP address given by your Wi-Fi router can change from time to time. This is annoying as you have to find out the new IP address allocated to your Raspberry Pi. Without knowledge of the IP address, you cannot log in using SSH or VNC.

In this section, you will learn how to fix your IP address so that it does not change between reboots. The steps are as follows:

- Log in to your Raspberry Pi 5 via Putty

- Check whether DHCP is active on your Raspberry Pi (it should normally be active):

  pi@raspberrypi:~ $ **sudo service dhcpcd status**

  If DHCP is not active, activate it by entering the following commands:

  pi@raspberrypi:~ $ **sudo service dhcpcd start**
  pi@raspberrypi:~ $ **sudo systemctl enable dhcpcd**

- Find the IP address currently allocated to you by entering the command **ifconfig** or **hostname – I** (Figure 2.12). In this example, the IP address was: 192.168.1.251. We can use this IP address as our fixed address, since no other device on the network is currently using it.

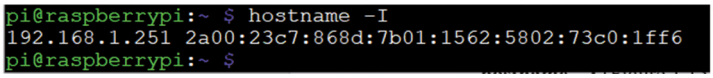

Figure 2.12 Find the IP address using the command hostname -I

- Find the IP address of your router by entering the command **ip r** (Figure 2.13). In this example, the IP address was: 192.168.1.254

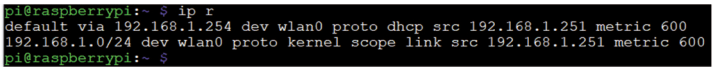

Figure 2.13 Find the IP address of your router.

- Find the IP address of your DNS by entering the following command (Figure 2.14). This is usually the same as your router address:

  pi@raspberrypi:~ $ **grep "nameserver" /etc/resolv.conf**

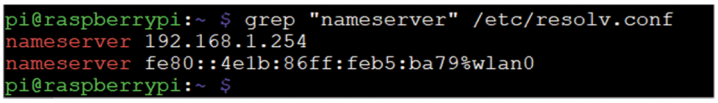

Figure 2.14 Find the DNS address.

- Edit file **/etc/dhcpcd.conf** by entering the command:

  pi@raspberrypi:~ $ **nano /etc/dhcpcd.conf**

- Add the following lines to the bottom of the file (these will be different for your router). If these lines already exist, remove the comment character '#' at the beginning of the lines and change the lines as follows (you may notice that **eth0** for Ethernet is listed):

```
interface wlan0
static_routers=192.168.1.254
static domain_name_servers=192.168.1.254
        static ip_address=192.168.1.251/24
```

- Save the file by entering **CTRL + X** followed by **Y** and reboot your Raspberry Pi
- In this example, the Raspberry Pi should reboot with the static IP address: 192.168.1.251

2.7 Enabling Bluetooth

In this section, you will see how to enable the Bluetooth on your Raspberry Pi 5 so that it can communicate with other Bluetooth devices. The steps are given below:

- Enable the Bluetooth on your other device
- Click on the Bluetooth icon on your Raspberry Pi 5 at the top right-hand side, and select **Make Discoverable.** You should see the Bluetooth icon flashing
- Select 'raspberrypi' in the Bluetooth menu on your other device
- Accept the pairing request on your Raspberry Pi 5
- You should now see the message **Connected Successfully** on your Raspberry Pi 5 and you can exchange files between your other device and the Raspberry Pi computer.

2.8 Connecting the Raspberry Pi 5 to a wired network

You may want to connect your Raspberry Pi 5 to a network through an Ethernet cable. The steps are as follows:

Step 1: Connect a network cable between your Raspberry Pi 5 and your Wi-Fi router.

Step 2: Connect the keyboard, mouse and monitor to your Raspberry Pi and power up as normal

Step 3: Log in to the system by entering your username and password

Step 4: Providing your network hub supports DHCP (nearly all network routers support DHCP), you will be connected automatically to the network and will be assigned a unique IP address within your network. Note that DHCP assigns IP addresses to newly connected devices.

Step 5: Check to find out the IP address assigned to your Raspberry Pi 5 by the network router. Enter the command **ifconfig** as described earlier

2.8.1 Unable to connect to a wired network

If you find out that you are not assigned an IP address by the DHCP server, possible causes are:

- Your network cable is faulty
- The network hub does not support DHCP
- DHCP is not enabled on your Raspberry Pi, i.e. it may have been configured for a fixed IP address

In most cases, it is very unlikely that the network cable is faulty. Also, most network hubs support the DHCP protocol. If you are having problems with the network, it is possible that your Raspberry Pi is not configured to accept DHCP issued addresses. The Raspberry Pi is normally configured to accept DHCP addresses, but it is possible that you have changed the configuration somehow.

To resolve the wired network connectivity problem, follow the steps given below:

Step 1: find out whether your Raspberry Pi is configured for DHCP or fixed IP addresses. Enter the following command:

```
pi@raspberrypi ~$ cat /etc/network/interfaces
```

If your Raspberry Pi is configured to use the DHCP protocol (which is normally the default configuration), the word **dhcp** should appear at the end of the following line:

```
iface eth0 inet dhcp
```

If, on the other hand, your Raspberry Pi is configured to use static addresses, then you should see the word **static** at the end of the following line:

```
iface eth0 inet static
```

Step 2: To use the DHCP protocol, edit file **interfaces** (e.g. using the **nano** text editor) and change the word **static** to **dhcp**. It is recommended to make a backup copy of the file interfaces before you change it:

```
pi@raspberrypi ~$ sudo cp /etc/network/interfaces /etc/network/int.bac
```

You should now restart your Raspberry Pi and an IP address will probably be assigned to your device.

Step 3: To use static addressing, make sure that the word **static** appears as shown above. If not, edit file **interfaces** and change **dhcp** to **static**

Step 4: You need to edit and add the required unique IP address, subnet mask and gateway addresses to file **interfaces** as in the following example (this example assumes that

the required fixed IP address is 192.168.1.251, the subnet mask used in the network is 255.255.255.0, and the gateway address is 192.168.1.1):

```
iface eth0 inet static
address 192.168.1.251
netmask 255.255.255.0
gateway 192.168.1.1
```

Save the changes and exit the editor. If you are using the **nano** editor, exit by pressing Ctrl+X, then enter Y to save the changes, and enter the filename to write to as **/etc/network/interfaces**.

Restart your Raspberry Pi 5.

2.9 Installing the Raspberry Pi 5 Bookworm operating system on a blank microSD card

If you have a pre-installed Raspberry Pi operating system Bookworm on a microSD card, then you can start using it as described earlier in this chapter. In this section, you will learn how to install the latest Bookworm operating system on a microSD card if you do not have a pre-installed card.

The steps are as follows:

- Insert a microSD card into your PC. You may need to use an SD card adapter
- Go to the website: https://www.raspberrypi.com/software/
- Click to download the **Raspberry Pi Imager.** At the time of writing this book, this file was called: **imager_1.7.5.exe**
- Double click to start the imager program and click to install it
- Click **Finish** to run the imager
- Click **Operating System** and select the operating system at the top of the list as: **Raspberry Pi OS (64-bit)**. See Figure 2.15

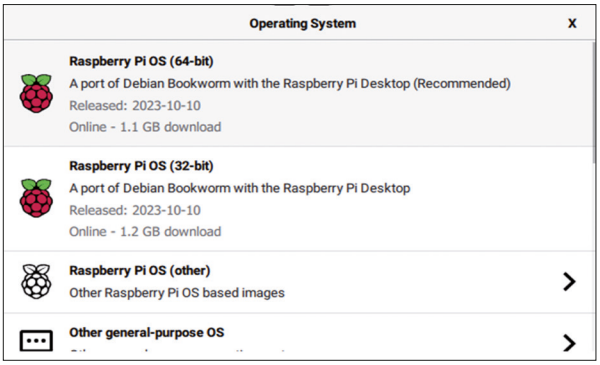

Figure 2.15 Select the operating system

- Click **Storage** and select the SD card storage
- Click to open the settings (gear shape)
- Click to enable SSH
- Click to enable password authentication
- Set username and password
- Click to **Configure wireless LAN**
- Click **Save**
- Click **Write** to write the operating system to the microSD card
- Wait until writing and verifying are finished (Figure 2.16)

- Remove the microSD card and insert into your Raspberry Pi 5

If you have a monitor and keyboard, you can log in to your Raspberry Pi 5 directly and start using it. Otherwise, find the IP address of your Raspberry Pi 5 (e.g. from your router, or there are many apps for smartphones, such as **who's on my wifi** that shows all the devices connected to your router with their IP addresses). Then log in to your Raspberry Pi 5 and start using it.

Chapter 3 • Using The Console Commands

3.1 Overview

Raspberry Pi is based on a version of the Linux operating system. Linux is one of the most popular operating systems in use today. Linux is very similar to other operating systems, such as Windows and UNIX. Linux is an open operating system based on UNIX and has been developed collaboratively by many companies since 1991. In general, Linux is harder to manage than some other operating systems like Windows, but offers more flexibility and configuration options. There are several popular versions of the Linux operating system, such as Debian, Ubuntu, Red Hat, Fedora and so on.

Linux commands are text-based. In this chapter, you will be looking at some of the useful Linux commands and see how you can manage your Raspberry Pi using these commands.

When you apply power to your Raspberry Pi 5, the Linux command line (or the Linux shell, or Console commands) is the first thing you see, and it is where you can enter operating system commands.

3.2 The Command Prompt

Assuming your username is **pi**, after you log in to Raspberry Pi 5, you will see the following prompt displayed where the system waits for you to enter a command:

```
pi@raspberrypi: ~$
```

Here, **pi** is the name of the user who is logged in.
raspberrypi is the name of the computer, used to identify it when connecting over the network.

~ character indicates that you are currently in your default directory.

3.3 Useful Console commands

In this section, you will be learning some of the useful Console commands, where examples will be given for each command. **In this chapter, commands entered by the user are shown in bold for clarity**. Also, it is important to remind you that all the commands must be terminated by the Enter key.

3.3.1 System and user information

These commands are useful as they tell you information about the system. Command **cat /proc/cpuinfo** displays information about the processor (command **cat** displays the contents of a file. In this example, the contents of file **/proc/cpuinfo** is displayed). Since there are four cores in the Raspberry Pi 5, the display is in four sections. Figure 3.1 shows an example display, where only part of the display is shown here.

```
pi@raspberrypi:~ $ cat /proc/cpuinfo
processor       : 0
BogoMIPS        : 108.00
Features        : fp asimd evtstrm aes pmull sha1 sha2 crc32 atomics fphp
p cpuid asimdrdm lrcpc dcpop asimddp
CPU implementer : 0x41
CPU architecture: 8
CPU variant     : 0x4
CPU part        : 0xd0b
CPU revision    : 1

processor       : 1
BogoMIPS        : 108.00
Features        : fp asimd evtstrm aes pmull sha1 sha2 crc32 atomics fphp
p cpuid asimdrdm lrcpc dcpop asimddp
CPU implementer : 0x41
CPU architecture: 8
CPU variant     : 0x4
CPU part        : 0xd0b
CPU revision    : 1

processor       : 2
BogoMIPS        : 108.00
```

Figure 3.1 Command: cat /proc/cpuinfo

Command **uname –s** displays the operating system kernel name, which is Linux. Command **uname –a** displays complete detailed information about the kernel and the operating system. An example is shown in Figure 3.2.

```
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 6.1.0-rpi4-rpi-2712 #1 SMP PREEMPT Debian 1:6.1.54-1+rpt1
3-09-27) aarch64 GNU/Linux
pi@raspberrypi:~ $
```

Figure 3.2 Command: uname – a

Command **cat /proc/meminfo** displays information about the memory on your Raspberry Pi. Information such as the total memory and free memory at the time of issuing the command are displayed. Figure 3.3 shows an example, where only part of the display is shown here.

```
pi@raspberrypi:~ $ cat /proc/meminfo
MemTotal:        8246848 kB
MemFree:         7792320 kB
MemAvailable:    7993952 kB
Buffers:           21552 kB
Cached:           246240 kB
SwapCached:            0 kB
Active:           280096 kB
Inactive:          64848 kB
Active(anon):      77008 kB
Inactive(anon):     4112 kB
Active(file):     203088 kB
Inactive(file):    60736 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:        102368 kB
SwapFree:         102368 kB
Zswap:                 0 kB
Zswapped:              0 kB
Dirty:                 0 kB
Writeback:             0 kB
AnonPages:         77232 kB
Mapped:            70880 kB
```

Figure 3.3 Command: cat /proc/meminfo

Command **whoami** displays the name of the current user. In this case, **pi** is displayed as the current user.

A new user can be added to your Raspberry Pi 5 using the command **useradd**. In the example in Figure 3.5, a user called **John** is added. A password for the new user can be added using the **passwd** command followed by the username. In Figure 3.4, the password for user John is set to **mypassword** (not displayed for security reasons). Notice that both the **useradd** and **passwd** are privileged commands, and the keyword **sudo** must be entered before these commands. Notice that the **–m** option creates a home directory for the new user.

```
pi@raspberrypi:~ $ sudo useradd -m John
pi@raspberrypi:~ $ sudo passwd John
New password:
Retype new password:
passwd: password updated successfully
pi@raspberrypi:~ $
```

Figure 3.4 Commands: useradd and passwd

You can log in to the new user account by specifying the username and the password as shown in Figure 3.5. You can type command **exit** to log out from the new account.

```
pi@raspberrypi:~ $ su John
Password:
John@raspberrypi:/home/pi $ exit
exit
pi@raspberrypi:~ $
```

Figure 3.5 Logging into a new account

Command **sudo apt-get upgrade** is used to upgrade all the software packages on the system.

3.3.2 The Raspberry Pi 5 directory structure

The Raspberry Pi 5 directory structure consists of a single root directory, with directories and subdirectories under the root. Different types of operating system programs and application programs are stored in different directories and subdirectories.

Figure 3.6 shows part of the Raspberry Pi 5 directory structure. Notice that the root directory is identified by the '**/**' symbol. Under the root we have directories named such as bin, boot, dev, etc, home, lib, lost+found, media, mnt, opt, proc, and many more. The important directory as far as the users are concerned is the **home** directory. The **home** directory contains subdirectories for each user of the system. In the example in Figure 3.7, **pi** is the subdirectory for user **pi**. In a new system, this subdirectory contains two subdirectories called **Desktop** and **python_games**.

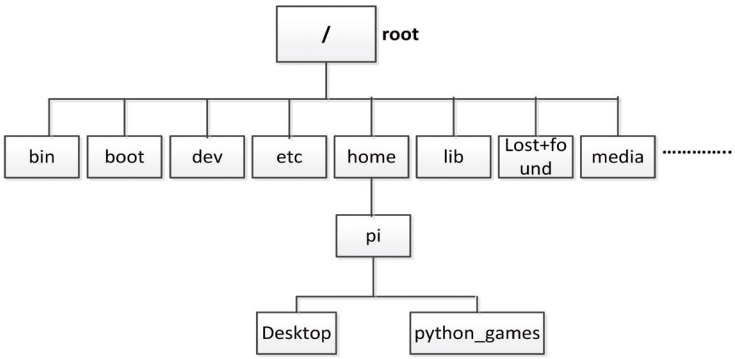

Figure 3.6 Raspberry Pi 5 directory structure (only part of it is shown)

Some useful directory commands are given below. Command **pwd** displays the user home directory:

```
pi@raspberrypi: ~$ pwd
/home/pi
pi@raspberry: ~$
```

To show the directory structure, enter the command **ls /** (Figure 3.7):

```
pi@raspberrypi:~ $ ls /
bin   etc         initrd.img.old  media  proc  sbin  tmp  vmlinuz
boot  home        lib             mnt    root  srv   usr  vmlinuz.old
dev   initrd.img  lost+found      opt    run   sys   var
pi@raspberrypi:~ $
```

Figure 3.7 Directory structure

To show the subdirectories and files in your working directory, enter **ls**:

```
pi@raspberrypi: ~$ ls
Bookshelf Documents Music Public Videos
Desktop Downloads Pictures Templates
pi@raspberrypi: ~$
```

Notice that the subdirectories are displayed in blue colour and files in black colour.

The **ls** command can take a number of arguments. Some examples are given below.
To display the subdirectories and files in a single row:

```
pi@raspberrypi: ~$ ls -1
Bookshelf
Desktop
Documents
Downloads
Music
```

```
Pictures
Public
Templates
Videos
pi@raspberrypi: ~$
```

To display the file types, enter the following command. Note that directories have a '/' after their names, and executable files have a '*' character after their names:

```
pi@raspberrypi: ~$ ls –F
Bookshelf/ Documents/ Music/ Public/ Videos/
Desktop/ Downloads/ Pictures/ Templates/
pi@raspberrypi: ~$
```

To list the results, separated by commas:

```
pi@raspberrypi: ~$ ls –m
```

Bookshelf, Desktop, Documents, Downloads, Music, Pictures, Public, Templates, Videos

```
pi@raspberrypi: ~$
```

You can mix the arguments as shown in Figure 3.8.

Figure 3.8 Mixing the arguments

Subdirectories are created using the command **mkdir** followed by the name of the subdirectory (Figure 3.9)

Figure 3.9 Creating a subdirectory

Command **find** is used to search the whole system for a file and outputs a list of all directories that contain the file. For example, the command **find / -name myfile.txt** searches the whole system for the file **myfile.txt.**

File Permissions

One of the important arguments used with the **ls** command is **-l** (lower case letter l) which displays the file permissions, file sizes, and when they were last modified. In the example below, each line relates to one directory or file. Reading from right to left, the name of the

directory or the file is on the right-hand side. The date the directory or file was created is on the left-hand side of its name. Next comes the size, given in bytes. The characters at the beginning of each line are about permissions, i.e. who is allowed to use or modify the file or the directory.

The permissions are divided into 3 categories:

- What the user (or owner, or creator) can do – called USER
- What the group owner (people in the same group) can do - GROUP
- What everyone else can do – called WORLD

The first word **pi** in the example in Figure 3.10 shows who the user of the file (or directory) is, and the second word **pi** shows the group name that owns the file. In this example, both the user and the group names are **pi**.

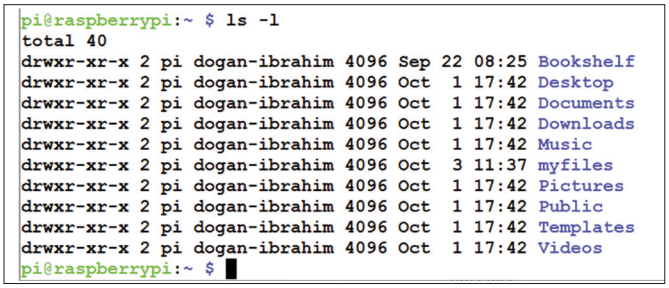

Figure 3.10 File permissions example

The permissions can be analysed by breaking down the characters into four chunks for: File type, User, Group, World. The first character for a file is '-' and for a directory, it is 'd'. Next come the permissions for the User, Group and World. The permissions are as follows:

- Read permission (r): the permission to open and read a file or to list a directory
- Write permission (w): the permission to modify a file, or to delete or create a file in a directory
- Execute permission (x): the permission to execute the file (applies to executable files), or to enter a directory

The three letters **rwx** are used as a group and if there is no permission assigned then a '-' character is used.

As an example, considering the **Music** directory, we have the following permission codes:

drwxr-xr-x which translates to:
d: it is a directory
rwx: user (owner) can read, write, and execute
r-x: group can read and execute, but cannot write (e.g. create or delete)
r-x: world (everyone else) can read and execute, but cannot write

The **chmod** command is used to change the file permissions. Before going into details of how to change the permissions, let us look and see what arguments are available in **chmod** for changing the file permissions.

The available arguments for changing file permissions are given below. We can use these arguments to add/remove permissions or to explicitly set permissions. It is important to realize that if we explicitly set permissions, then any unspecified permissions in the command will be revoked:

u:	user (or owner)
g:	group
o:	other (world)
a:	all
+:	add
-:	remove
=:	set
r:	read
w:	write
x:	execute

To change the permissions of a file we type the **chmod** command, followed by one of the letters 'u', 'g', 'o', or 'a' to select the people, followed by the '+', '-' or '=' to select the type of change, and finally followed by the filename. In this example, a file with the name **mytestfile.txt** was created in the home directory for demonstration purposes (See Figure 3.11). In this example, the file **mytestfile.txt** has the user read and write permissions.

```
pi@raspberrypi:~ $ ls -l
total 44
drwxr-xr-x 2 pi dogan-ibrahim 4096 Sep 22 08:25 Bookshelf
drwxr-xr-x 2 pi dogan-ibrahim 4096 Oct  1 17:42 Desktop
drwxr-xr-x 2 pi dogan-ibrahim 4096 Oct  1 17:42 Documents
drwxr-xr-x 2 pi dogan-ibrahim 4096 Oct  1 17:42 Downloads
drwxr-xr-x 2 pi dogan-ibrahim 4096 Oct  1 17:42 Music
drwxr-xr-x 2 pi dogan-ibrahim 4096 Oct  3 11:37 myfiles
-rw-r--r-- 1 pi dogan-ibrahim   15 Oct  3 11:46 mytestfile.txt
drwxr-xr-x 2 pi dogan-ibrahim 4096 Oct  1 17:42 Pictures
drwxr-xr-x 2 pi dogan-ibrahim 4096 Oct  1 17:42 Public
drwxr-xr-x 2 pi dogan-ibrahim 4096 Oct  1 17:42 Templates
drwxr-xr-x 2 pi dogan-ibrahim 4096 Oct  1 17:42 Videos
pi@raspberrypi:~ $
```

Figure 3.11 File permissions

We will be changing the permissions so that the user does not have read permission on this file:

```
pi@raspberrypi: ~$ chmod u-r mytestfile.txt
pi@raspberrypi: ~$ ls –lh
```

The result is shown in Figure 3.12.

```
pi@raspberrypi:~ $ chmod u-r mytestfile.txt
pi@raspberrypi:~ $ ls -lh
total 44K
drwxr-xr-x 2 pi dogan-ibrahim 4.0K Sep 22 08:25 Bookshelf
drwxr-xr-x 2 pi dogan-ibrahim 4.0K Oct  1 17:42 Desktop
drwxr-xr-x 2 pi dogan-ibrahim 4.0K Oct  1 17:42 Documents
drwxr-xr-x 2 pi dogan-ibrahim 4.0K Oct  1 17:42 Downloads
drwxr-xr-x 2 pi dogan-ibrahim 4.0K Oct  1 17:42 Music
drwxr-xr-x 2 pi dogan-ibrahim 4.0K Oct  3 11:37 myfiles
--w-r--r-- 1 pi dogan-ibrahim   15 Oct  3 11:46 mytestfile.txt
drwxr-xr-x 2 pi dogan-ibrahim 4.0K Oct  1 17:42 Pictures
drwxr-xr-x 2 pi dogan-ibrahim 4.0K Oct  1 17:42 Public
drwxr-xr-x 2 pi dogan-ibrahim 4.0K Oct  1 17:42 Templates
drwxr-xr-x 2 pi dogan-ibrahim 4.0K Oct  1 17:42 Videos
pi@raspberrypi:~ $
```

Figure 3.12 File permissions of **mytestfile.txt**

Notice that if you now try to display the contents of the file **mytestfile.txt** using the **cat** command, you will get an error message:

```
pi@raspberrypi: ~$ cat mytestfile.txt
cat: mytestfile.txt: Permission denied
pi@raspberrypi: ~$
```

All the permissions can be removed from a file by the following command:

```
pi@raspberrypi: ~$ chmod a=  mytestfile.txt
```

In the following example, **rwx** user permissions are given to file **mytestfile.txt**:

```
pi@raspberrypi: ~$ chmod u+rwx mytestfile.txt
```

Figure 3.13 shows the new permissions of file mytestfile.txt.

```
pi@raspberrypi:~ $ chmod u+rwx mytestfile.txt
pi@raspberrypi:~ $ ls -l
total 44
drwxr-xr-x 2 pi dogan-ibrahim 4096 Sep 22 08:25 Bookshelf
drwxr-xr-x 2 pi dogan-ibrahim 4096 Oct  1 17:42 Desktop
drwxr-xr-x 2 pi dogan-ibrahim 4096 Oct  1 17:42 Documents
drwxr-xr-x 2 pi dogan-ibrahim 4096 Oct  1 17:42 Downloads
drwxr-xr-x 2 pi dogan-ibrahim 4096 Oct  1 17:42 Music
drwxr-xr-x 2 pi dogan-ibrahim 4096 Oct  3 11:37 myfiles
-rwx------ 1 pi dogan-ibrahim   15 Oct  3 11:46 mytestfile.txt
drwxr-xr-x 2 pi dogan-ibrahim 4096 Oct  1 17:42 Pictures
drwxr-xr-x 2 pi dogan-ibrahim 4096 Oct  1 17:42 Public
drwxr-xr-x 2 pi dogan-ibrahim 4096 Oct  1 17:42 Templates
drwxr-xr-x 2 pi dogan-ibrahim 4096 Oct  1 17:42 Videos
pi@raspberrypi:~ $
```

Figure 3.13 New permissions of file **mytestfile.txt**

To change our working directory, the command **cd** is used. In the following example, we change our working directory to **Music**:

```
pi@raspberrypi: ~$ cd /home/pi/Music
pi@raspberrypi: ~/Music $
```

To go up one directory level, i.e. to our default working directory:

```
pi@raspberrypi: ~/Music $ cd..
pi@raspberrypi: ~$
```

To change your working directory to **Music**, you can also enter the command:

```
pi@raspberrypi: ~$ cd ~/Music
pi@raspberrypi: ~/myfiles $
```

To go back to the default working directory, you can enter:

```
pi@raspberrypi: ~/Music $ cd ~
pi@raspberrypi: ~$
```

To find out more information about a file, you can use the **file** command. For example:

```
pi@raspberrypi: ~$ file mytestfiile.txt
mytestfile.txt: ASCII text
pi@raspberrypi: ~$
```

The **–R** argument of the command **ls** lists all the files in all the subdirectories of the current working directory. An example is given below (only part of the display is shown). Notice here in Figure 3.14 that subdirectory **Bookshelf** contains file **BeginnersGuide-4thEd-Eng_v2.pdf**

```
pi@raspberrypi:~ $ ls -R
.:
Bookshelf  Documents  Music     mytestfile.txt  Public     Videos
Desktop    Downloads  myfiles   Pictures        Templates

./Bookshelf:
BeginnersGuide-4thEd-Eng_v2.pdf

./Desktop:

./Documents:

./Downloads:
```

Figure 3.14 Command: **ls –R**

To display information on how to use a command, you can use the **man** command. As an example, to get help on using the **mkdir** command:

```
pi@raspberrypi: ~$ man mkdir
MKDIR(1)

NAME
        Mkdir - make directories

SYNOPSIS
```

```
               Mkdir [OPTION]…DIRECTORY…

       DESCRIPTION
               Create the DIRECTORY(ies), if they do not already exist.

               Mandatory arguments for long options are mandatory for short
options

               -m, --mode=MODE
                              Set file mode (as in chmod), not a=rwx - umask
       ------------------------------------------------------------------------
       ------------------------------------------------------------------------
```

Enter **q** to exit the man display.

Help
The **man** command usually gives several pages of information on how to use a command. You can type **q** to exit the **man** command and return to the operating system prompt.

The **less** command can be used to display a long listing one page at a time. Using the up and down arrow keys, we can move between pages. An example is given below. Type **q** to exit:

pi@raspberrypi: ~$ **man ls | less**
<display of help on using the ls command>
pi@raspberrypi: ~$

Date and Time
To display the current date and time, the **date** command is used.

Copying a File
To make a copy of a file, use the command **cp**. In the following example, a copy of the file **mytestfile.txt** is made, and the new file is given the name **test.txt**:

pi@raspberrypi: ~$ **cp mytestfile.txt test.txt**
pi@raspberrypi: ~$

Wildcards
You can use wildcard characters to select multiple files with similar characteristics. e.g. files having the same file-extension names. The ***** character is used to match any number of characters. Similarly, the **?** character is used to match any single character. In the example below, all the files with extensions **.txt** are listed:

pi@raspberrypi: ~$ **ls *.txt**
mytestfile.txt test.txt
pi@raspberrypi: ~$

The wildcard characters [a-z] can be used to match any single character in the specified character range. An example is given below which matches any files that start with the letters 'o', 'p', 'q', 'r', 's', and 't', and with the **.txt** extension:

```
pi@raspberrypi: ~$ ls [o-t]*.txt
test.txt
pi@raspberrypi: ~$
```

Renaming a File
You can rename a file using the **mv** command. In the example below, the name of file **test.txt** is changed to **test2.txt**:

```
pi@raspberrypi: ~$ mv test.txt test2.txt
pi@raspberrypi: ~$
```

Deleting a File
The command **rm** can be used to remove (delete) a file. In the example below, the file **test2.txt** is deleted:

```
pi@raspberrypi: ~$ rm test2.txt
pi@raspberrypi: ~$
```

The argument **–v** can be used to display a message when a file is removed. Also, the **–i** argument asks for confirmation before a file is removed. In general, the two arguments are used together as **–vi**. An example is given below:

```
pi@raspberrypi: ~$ rm –vi test2.txt
rm: remove regular file 'test2.txt'? y
removed 'test2.txt'
pi@raspberrypi: ~$
```

Sorting a file
The command sort displays the contents of a file in ascending order. The general format of this command is:

sort <options> <filename>

Valid options are:

-u	removes duplicates from the output
-r	sorts the output in descending order
-o	writes the sorted output to a file

Word count
Command **wc** <filename> displays the word count in a file

File differences
Command **diff** <file1> <file2) displays the differences between two files line by line

Removing a Directory
A directory can be removed using the **rmdir** command:

```
pi@raspberrypi: ~$ rmdir Music
pi@raspberrypi: ~$
```

Re-directing the Output
The greater sign **>** can be used to redirect the output of a command to a file. For example, we can redirect the output of the **ls** command to a file called **lstest.txt**:

```
pi@raspberrypi: ~$ ls > lstest.txt
pi@raspberrypi: ~$
```

The **cat** command can be used to display the contents of a file:

```
pi@raspberrypi: ~$ cat mytestfile.txt
This is a file
This is line 2
pi@raspberrypi: ~$
```

Using two greater signs '**>>**' adds to the end of a file.

Writing to the Screen or to a File
The **echo** command can be used to write to the screen. It can be used to perform simple mathematical operations if the numbers and the operation are enclosed in two brackets, preceded by a $ character:

```
pi@raspberrypi: ~$ echo $((5*6))
30
pi@raspberrypi: ~$
```

The echo command can also be used to write a line of text to a file. An example is shown below:

```
pi@raspberrypi: ~$ echo a line of text > lin.dat
pi@raspberrypi: ~$ cat lin.dat
a line of text
pi@raspberrypi: ~$
```

Matching a String
The grep command can be used to match a string in a file. An example is given below, assuming that the file lin.dat contains sting a line of text. Notice that the matched word is shown in bold:

```
pi@raspberrypi: ~$ grep line lin.dat
a line of text
pi@raspberrypi: ~$
```

Head and Tail Commands
The head command can be used to display the first 10 lines of a file. The format of this command is as follows:

```
pi@raspberrypi: ~$ head mytestfile.txt
……..
……..
pi@raspberrypi: ~$
```

Similarly, the tail command is used to display the last 10 lines of a file. The format of this command is as follows:

```
pi@raspberrypi: ~$ tail mytestfile.txt
……..
……..
pi@raspberrypi: ~$
```

The **which** command displays the location of an executable program. For example, the location of the python program can be found as follows:

```
pi@raspberrypi: ~$ which python
/usr/bin/python
pi@raspberrypi: ~$
```

Super User Commands
Some of the commands are privileged and only the authorized persons can use them. Inserting the word **sudo** at the beginning of a command gives us the authority to use the command without having to log in as an authorized user.

What software is installed on my Raspberry Pi 5
To find out what software is installed on your Raspberry Pi 5, enter the following command. You should get several pages of display:

```
pi@raspberrypi: ~$ dpkg –l
……..
……..
pi@raspberrypi: ~$
```

You can also find out if a certain software package is already installed on our computer. An example is given below which checks whether software called **xpdf** (PDF reader) is installed. In this example, **xpdf** is installed and the details of this software are displayed:

```
pi@raspberrypi: ~$ dpkg --s xpdf
Package: xpdf
Status: install ok installed
Priority: optional
Section: text
Installed-Size: 395
……..
……..
pi@raspberrypi: ~$
```

If the software is not installed, you get a message similar to the following (assuming we are checking to see if a software package called **bbgd** is installed):

```
pi@raspberrypi: ~$ dpkg –s bbgd
dpkg-query: package 'bbgd' is not installed and no information is available
……..
……..
pi@raspberrypi: ~$
```

3.3.3 Resource monitoring on the Raspberry Pi 5

System monitoring is an important topic for managing usage of your Raspberry Pi. One of the most useful system monitoring commands is the top, which displays the current usage of system resources and displays which processes are running and how much memory and CPU time they are consuming.

Figure 3.15 shows a typical system resource display obtained by entering the following command (only part of the display is shown, Enter **q** to exit):

```
pi@raspberrypi: ~$ top
pi@raspberrypi: ~$
```

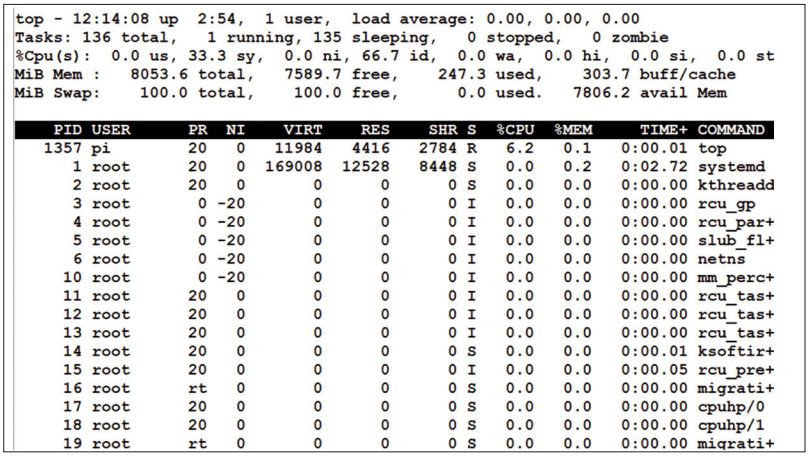

Figure 3.15 Typical system resource display

Some of the important points in Figure 3.15 are summarized below (for lines 1 to 5 of the display):

- There are a total of 138 processes in the system
- Currently, only one process is running, 1 process is sleeping, and 0 processes are stopped
- The percentage of CPU utilization is 0.0 'us' for user applications (us)
- The percentage of CPU utilization for system applications is 0.0 (sy)
- There are no processes requiring more or less priority (ni)
- 100% of the time the CPU is idle (id)
- There are no processes waiting for I/O completion (wa)
- There are no processes waiting for hardware interrupts (hi)
- There are no processes waiting for software interrupts (si)
- There is no time reserved for a hypervisor (st)
- The total usable memory is 8053 bytes, of which 247 bytes are in use, 7589 bytes are free, and 303 bytes are used by buffers/cache
- Line 5 displays the swap space usage

The process table gives the following information for all the processes loaded to the system:

- PID: the process ID number
- USER: owner of the process
- PR: priority of the process
- NI: the nice value of the process
- VIRT: the amount of virtual memory used by the process
- RES: size of the resident memory
- SHR: shared memory the process is using
- S: process status (sleeping, running, zombie)
- %CPU: the percentage of CPU consumed
- %MEM: percentage of RAM used
- TIME+: total CPU time the task used
- COMMAND: The actual name of the command

The command **htop** is similar to the **top** command, except it has more features and is more user-friendly.

The **ps** command can be used to list all the processes used by the current user. An example is shown in Figure 3.16.

```
pi@raspberrypi:~ $ ps
    PID TTY          TIME CMD
    971 pts/0    00:00:00 bash
   1372 pts/0    00:00:00 ps
pi@raspberrypi:~ $
```

Figure 3.16 Command: **ps**

Command **ps –ef** gives a lot more information about the processes running in the system.

Killing a process

There are many options for killing (or stopping) a process. A process can be killed by specifying its PID and using the following command:

pi@raspberrypi: ~$ **kill -9 <PID>**

Disk (microSD card) usage

The disk free command **df** can be used to display the disk usage statistics. An example is shown in Figure 3.17. option **–h** displays in human-readable form.

```
pi@raspberrypi:~ $ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            3.8G     0  3.8G   0% /dev
tmpfs           806M  3.6M  802M   1% /run
/dev/mmcblk0p2   29G  4.6G   23G  17% /
tmpfs           4.0G   32K  4.0G   1% /dev/shm
tmpfs           5.0M   48K  5.0M   1% /run/lock
/dev/mmcblk0p1  510M   73M  438M  15% /boot/firmware
tmpfs           806M  128K  806M   1% /run/user/1000
pi@raspberrypi:~ $
```

Figure 3.17 Command: **df -h**

Command **free** shows how much memory is used and the amount of free memory.

3.3.4 Shutting Down

Although you can disconnect the power supply from your Raspberry Pi 5 when you finish working with it, it is not recommended since there are many processes running on the system, and it is possible to corrupt the file system. It is much better to shut down the system in an orderly manner.

The following command will stop all the processes and make the file system safe, and then turn off the system safely:

pi@raspberrypi: ~$ **sudo halt**

The following command stops and then restarts the system:

pi@raspberrypi: ~$ **sudo reboot**

The system can also be shut down and then restarted after a time by entering the following command. Optionally, a shutdown message can be displayed if desired:

pi@raspberrypi: ~$ **shutdown –r <time> <message>**

To shutdown at 1:55 AM:

pi@raspberrypi: ~$ **sudo shutdown -h 01:55:**

Enter the following command to shut down now:

pi@raspberrypi: ~$ **sudo shutdown now**

Broadcast message from root @raspberrypi on pts/1 (Tue 2023-10-03 12:03:00 BST)

The system will power off now!

Note: Raspberry Pi 5 includes a power switch at its side. When the Raspberry Pi is ON, a single press brings the shutdown/logout menu. Another press triggers a safe shutdown, which is a standby with the Raspberry Pi consuming about 1.4 W. A press of the button will start up the Raspberry Pi 5.

3.3.5 Networking

Some useful networking commands are:

ifconfig: check the IP address of your Raspberry Pi

iwconfig: check which network the Raspberry Pi is using. An example is shown in Figure 3.18. Here, the SSID of the Wi-Fi adapter used is BTHub5-6SPN

```
pi@raspberrypi:~ $ iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

wlan0     IEEE 802.11  ESSID:"BTHub5-6SPN"
          Mode:Managed  Frequency:5.18 GHz  Access Point: 4C:1B:86:B5:BA:7B
          Bit Rate=433.3 Mb/s   Tx-Power=31 dBm
          Retry short limit:7   RTS thr:off   Fragment thr:off
          Power Management:on
          Link Quality=45/70  Signal level=-65 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:3  Invalid misc:0   Missed beacon:0

pi@raspberrypi:~ $
```

Figure 3.18 Command iwconfig

ping: used to test the availability of a network device. An example is shown in Figure 3.19

```
pi@raspberrypi:~ $ ping 192.168.1.251
PING 192.168.1.251 (192.168.1.251) 56(84) bytes of data.
64 bytes from 192.168.1.251: icmp_seq=1 ttl=64 time=0.033 ms
64 bytes from 192.168.1.251: icmp_seq=2 ttl=64 time=0.014 ms
64 bytes from 192.168.1.251: icmp_seq=3 ttl=64 time=0.011 ms
```

Figure 3.19 Command ping

wget: this command is used to download a file from the web and saves the file in the current directory.

hostname – I: shows the IP address of the Raspberry Pi

The command **vcgencmd measure_temp** displays the CPU temperature as shown in Figure 3.20.

```
pi@raspberrypi:~ $ vcgencmd measure_temp
temp=49.4'C
pi@raspberrypi:~ $
```

Figure 3.20 Displaying the CPU temperature

3.3.6 System information and other useful commands

The **uname** command is used to display system information. This command has the following options:

-a	Show all system information
-s	display the kernel name
-n	print the network node hostname
-r	print the kernel release
-v	print the kernel version number
-m	print the system hardware name
-p	print the processor type
-i	print the hardware platform type
-o	print the operating system type

Some examples are shown in Figure 3.21

```
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 6.1.0-rpi4-rpi-2712 #1 SMP PREEMPT Debian 1:6.1.54-1+rpt1
3-09-27) aarch64 GNU/Linux
pi@raspberrypi:~ $ uname -s
Linux
pi@raspberrypi:~ $ uname -n
raspberrypi
pi@raspberrypi:~ $ uname -r
6.1.0-rpi4-rpi-2712
```

Figure 3.21 The uname command

If you have executed many commands and want to use some of them again, but you cannot remember the command name, you can use the **history** command. An example is shown in Figure 3.22. To execute a command from the history, enter **!** followed by the command number. For example, to execute the **ls** command again, you can enter **!6** followed by the Enter key.

```
pi@raspberrypi:~ $ history
    1  ls
    2  sudo nano /etc/default/console-setup
    3  sudo /etc/init.d/console-setup restart
    4  sudo restart
    5  sudo reboot
    6  ls
    7  cat /etc/default/console-setup
    8  sudo dpkg-reconfigure console-setup
    9  ls
   10  ls music
```

Figure 3.22 The history command

The **clear** command is also useful, and it is used to clear the screen.

To install a package, use the command: **sudo apt install <package_name>**

The **&** operator allows you to run any command in the background so that you can use the terminal for other tasks. This operator must be added to the end of a command.

The **&&** operator allows you to run two or more commands at the same time. For example, command1 **&&** command2

Chapter 4 • Desktop GUI – Desktop Applications

4.1 Overview

In this chapter, you will learn how to access and use the desktop applications of your Raspberry Pi 5.

4.2 Desktop GUI Applications

If you have connected a monitor, a mouse, and a keyboard to your Raspberry Pi 5 computer, then you can access the desktop GUI by entering command **startx** in the command mode.

If you wish to access the Desktop GUI applications from a desktop or a laptop computer, then the required steps are as follows:

Step 1: Connect to your Raspberry Pi 5 using the Putty terminal emulator with the SSH services as explained in earlier chapters.

Step 2: Run the VNC server by entering the following command into your SSH window:

pi@raspberrypi ~ $ **vncserver :1**

Step 3: Run the **VNC Viewer** program on your computer. Enter the IP address of your Raspberry Pi 5 computer, followed by characters :1 to indicate that we are using port 1 (see Figure 4.1). Click the **Connect** button.

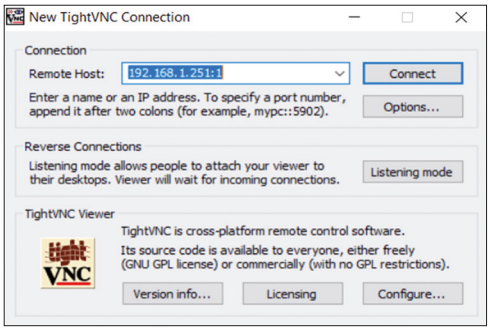

Figure 4.1 Enter the IP address of your Pi 5

You will be asked for your password that you created earlier. Enter the password, and you will see the Raspberry Pi 5 Desktop displayed (Figure 4.2, only the upper part of the screen is shown)

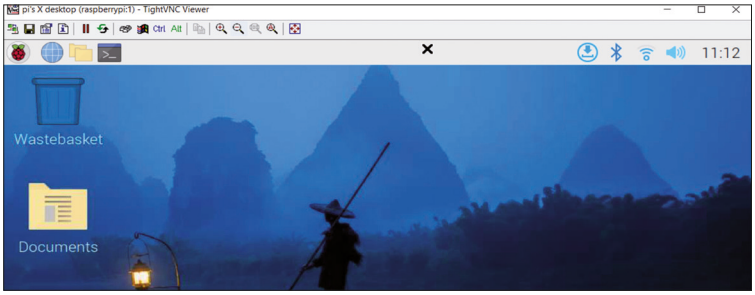

Figure 4.2 Raspberry Pi 5 Desktop

Assuming you are using a pre-installed micro SD card, at the top of the screen you have a number of shortcut icons. Below that, you will see four menu icons with the names:

Applications menu
Web Browser
File Manager
Terminal

On the top right-hand side of the screen, starting from the left, you have the following menus:

Updates
Bluetooth
Wi-Fi
Volume Control
Time

4.2.1 Applications Menu

Figure 4.3 shows the items under the Applications Menu.

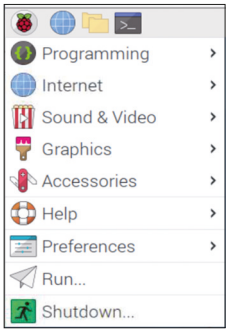

Figure 4.3 Items under the Applications Menu

Programming: This menu item includes a number of programming languages that you can use to program our Raspberry Pi 5. Figure 4.4 shows a list of the items in the Programming menu.

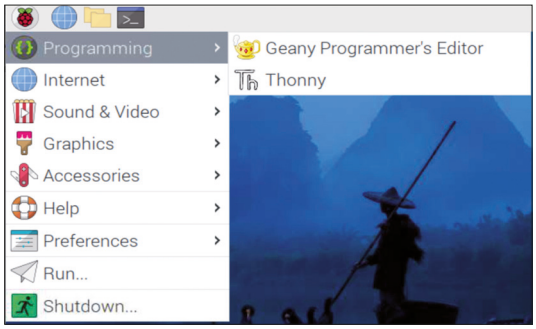

Figure 4.4 Items under the Programming menu

In this book, you will be using the **Thonny IDE** in most of your Python programs. Details about the Thonny IDE are given in a later chapter.

Web Browser: This menu item includes the **Chromium** and the **Firefox** web browsers.

Sound & Video: This menu item includes the video program **VLC Media Player**

Graphics: This menu item includes the **Image Viewer** program

Accessories: This menu item includes a number of useful programs, as shown in Figure 4.5. For example, the Calculator program can be used to perform simple and scientific calculations (Figure 4.6)

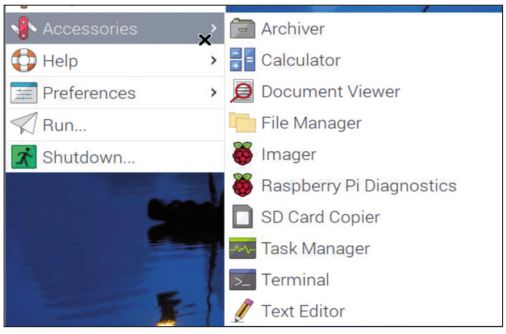

Figure 4.5 The Accessories menu

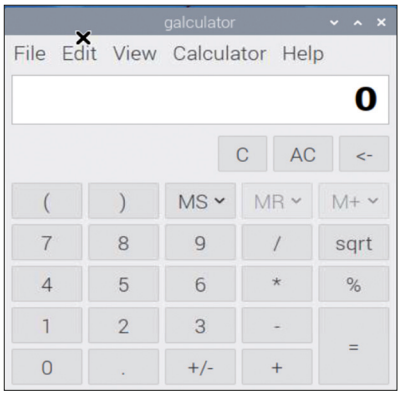

Figure 4.6 The Calculator program

Help: This menu provides help on various Raspberry Pi tools

Preferences: This menu is used for system and software settings, such as adding/removing software, print services, screen configuration, etc. (Figure 4.7)

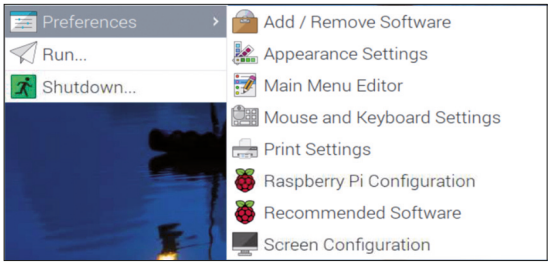

Figure 4.7 The Preferences menu

Run: This menu is used to run a program

Shutdown: Use this menu option to shut down your Raspberry Pi 5

4.2.2 Web browser

Click this menu option to start a web browser

4.2.3 File manager

This menu item is used for file handling and is similar to the File Explorer on Windows systems. Figure 4.8 shows the options under this menu item. With the File manager, you can create a file, copy/paste text, view the contents of a file, sort a file, find files, and several other file processing options.

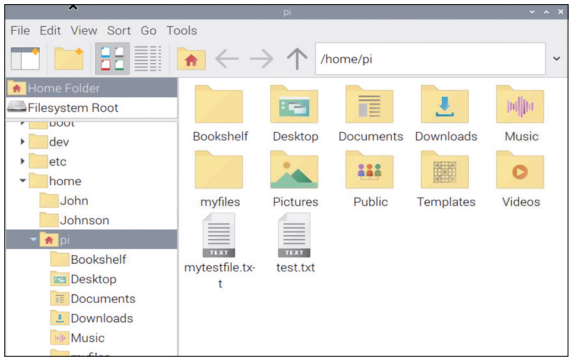

Figure 4.8 The File manager menu

4.2.4 Terminal

This menu item enables the command mode so that you can enter commands in this mode (Figure 4.9, only part of the screen is shown)

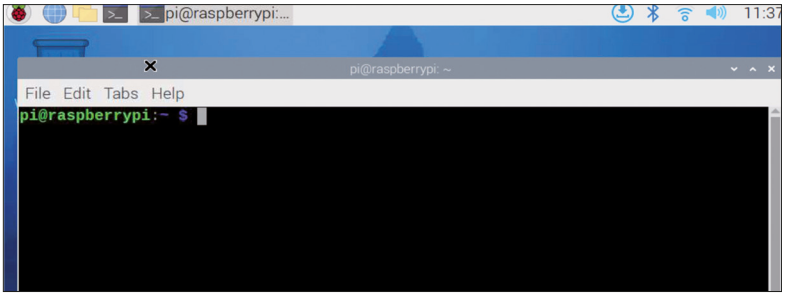

Figure 4.9 The Terminal menu

4.2.5 Manage Bluetooth devices

This menu item at the right-hand side of the screen enables you to enable the Bluetooth on your Raspberry Pi 5 and to pair with other Bluetooth devices

4.2.6 Wi-Fi

The next menu item to the Bluetooth is the Wi-Fi menu, which can be used to turn the Wi-Fi on and off, and to connect to a Wi-Fi router. When clicked, a list of the available Wi-Fi devices is given (see Figure 4.10).

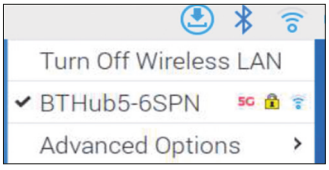

Figure 4.10 Wi-Fi menu

4.2.7 Volume control

This menu item is used to control the audio volume through a sliding bar.

4.2.8 Date and time

Next to the Volume control is the date and time menu, which shows the current system date and time.

Chapter 5 • Using a Text Editor in Console Mode

A text editor is used to create or modify the contents of a text file. There are many text editors available for the Linux operating system. Some popular ones are nano, vim, vi, and many more. In this chapter, we shall be looking at some of these text editors and see how to use them.

5.1 nano text editor

Start the **nano** text editor by entering the word **nano**, followed by the filename you wish to create or modify. An example is given below where a new file called **first.txt** is created:

```
pi@raspberrypi: ~ $ nano first.txt
```

You should see the editor screen as in Figure 5.1. The name of the file to be edited is written at the top middle part of the screen. The message 'New File' at the bottom of the screen shows that this is a newly created file. The shortcuts at the bottom of the screen are there to perform various editing functions. These shortcuts are accessed by pressing the Ctrl key together with another key. Some of the useful shortcuts are given below:

Ctrl+W: Search for a word
Ctrl+V: Move to the next page
Ctrl+Y: Move to the previous page
Ctrl+K: Cut the current row of txt
Ctrl+R: Read file
Ctrl+U: Paste the text you previously cut
Ctrl+J: Justify
Ctrl+\: Search and replace text
Ctrl+C: Display current column and row position
Ctrl+G: Get detailed help on using the nano
Ctrl+-: Go to the specified line and column position
Ctrl+O: Save (write out) the file currently open
Ctrl+X: Exit nano

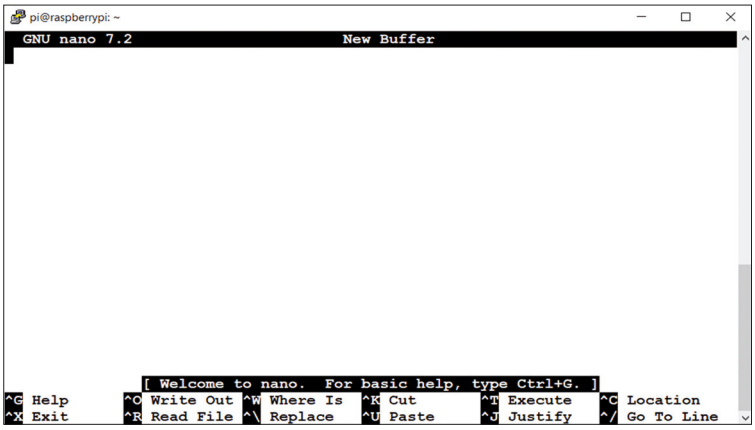

Figure 5.1 nano text editor screen

Now, type the following text as shown in Figure 5.2:

```
nano is a simple and yet powerful text editor.
This simple text example demonstrates how to use nano.
This is the last line of the example.
```

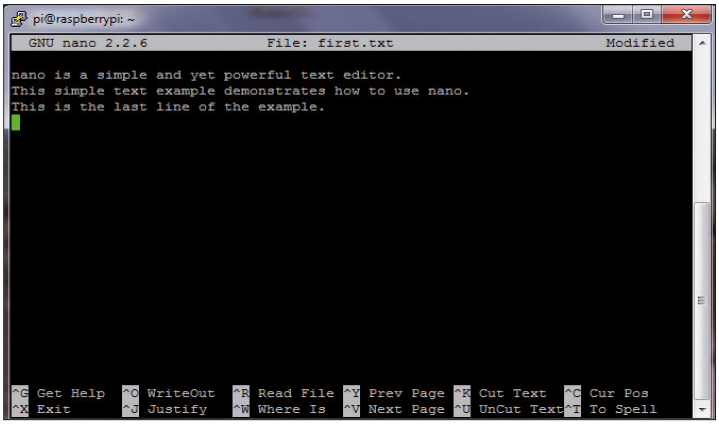

Figure 5.2 Sample text

The use of **nano** is now demonstrated with the following steps:

Step 1: Go to the beginning of the file by moving the cursor.

Step 2: Look for the word **simple** by pressing **Ctrl+W** and then typing **simple** in the window opened at the bottom left-hand corner of the screen. Press the Enter key. The cursor will be positioned on the word **simple** (see Figure 5.3).

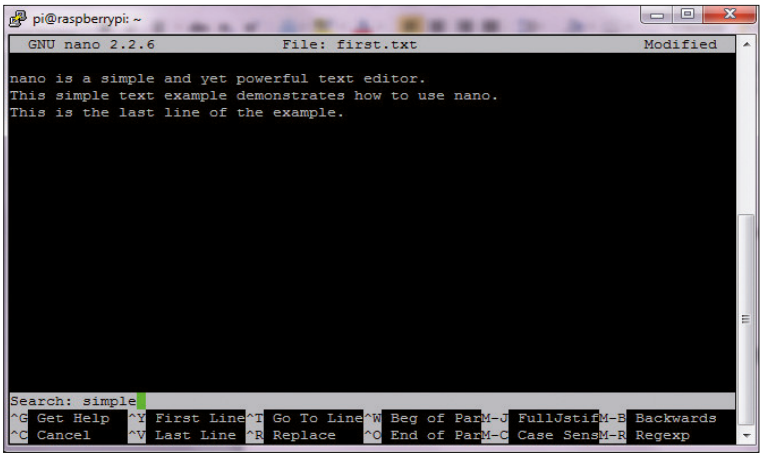

Figure 5.3 Searching the word **simple**

Step 3: Cut the first line by placing the cursor anywhere on the line and then pressing Ctrl+K. The first line will disappear, as in Figure 5.4.

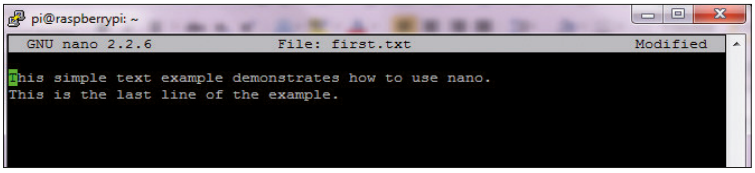

Figure 5.4 Cutting the first line

Step 4: Paste the line cut after the first line. Place the cursor on the second line and press **Ctrl+U** (see Figure 5.5).

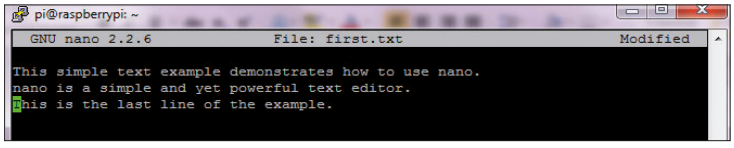

Figure 5.5 Paste the line cut previously

Step 5: Place cursor at the beginning of the word **simple** on the first row. Enter **Ctrl+C**. The row and column positions of this word will be displayed at the bottom of the screen (Figure 5.6).

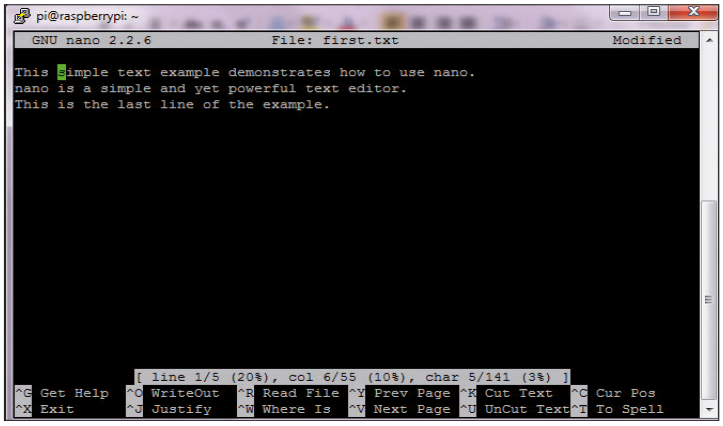

Figure 5.6 Displaying row and column position of a word

Step 6: Press **Ctrl+G** to display the help page as in Figure 5.7. Notice that the display is many pages long, and you can jump to the next pages by pressing **Ctrl+Y** or to the previous pages by pressing **Ctrl+V**. Press **Ctrl+X** to exit the help page.

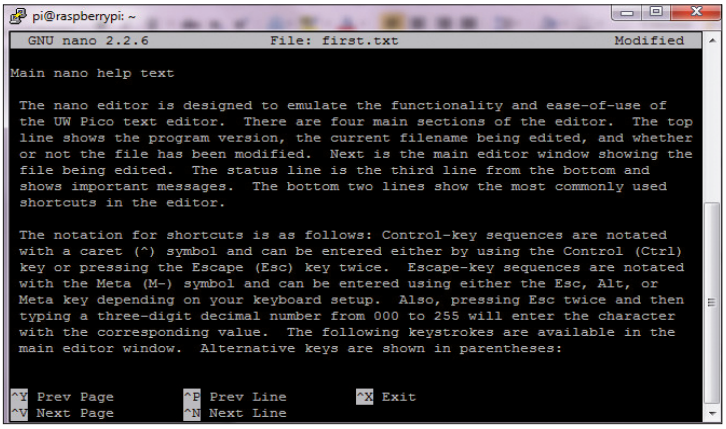

Figure 5.7 Displaying the help page

Step 7: Press **Ctrl+-** and enter line and column numbers as 2 and 5, followed by the Enter key, to move the cursor to line 2, column 5 (see Figure 5.8).

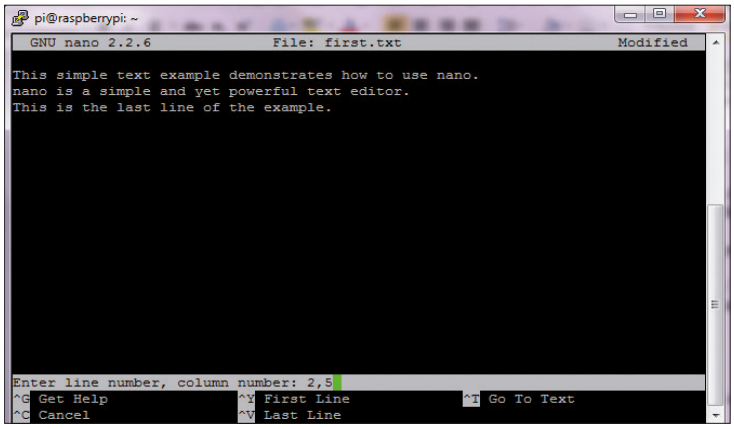

Figure 5.8 Moving to line 2, column 5

Step 8: Replace word **example** with word **file**. Press **Ctrl+** and type the first word as **example** (see Figure 5.9). Press Enter and then type the replacement word as **file**. Press Enter and accept the change by typing y.

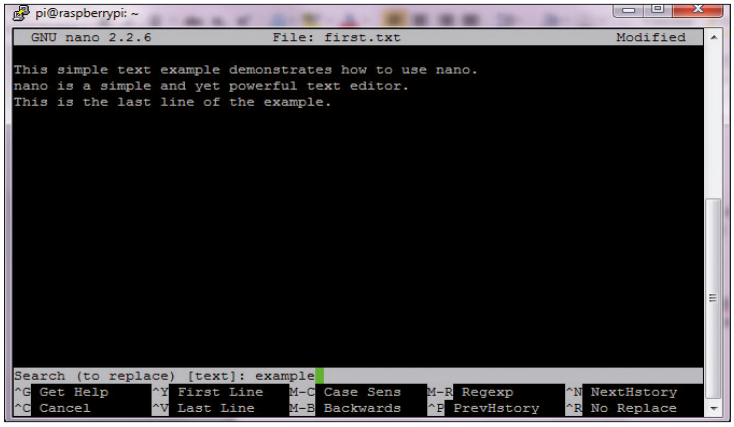

Figure 5.9 Replacing text

Step 9: Save the changes. Press **Ctrl+X** to exit the file. Type **Y** to accept the saving, then enter the filename to be written to, or simply press Enter to write to the existing file (**first.txt** in this example). The file will be saved in your current working directory.

Step 10: Display the contents of the file:

```
pi@raspberrypi: ~ $ cat firs.txt
This simple text file demonstrates how to use nano.
Nano is a simple and yet powerful text editor
This is the last line of the example.

pi@raspberrypi: ~ $
```

In summary, **nano** is a simple and yet powerful text editor, allowing us to create new text files or to edit existing files.

5.2 vi text editor

The **vi** text editor has been around for many years when it has been the standard Unix operating system default text editor. The **vi** editor is a fully featured, powerful text editor for doing many different tasks. The only problem with using **vi** is that it is not very user-friendly and learning may take some time. In this section, we shall be looking at the basic features of this editor and see how we can use it in simple editing applications.

Notice that you cannot use the keyboard arrow keys with the **vi** editor. Some of the useful **vi** editor commands are listed below:

```
ZZ   save changes and exit vi
:wq  save changes and exit vi
:q!  exit without saving changes

h    move cursor left (backwards)
j    move cursor down
k    move cursor up
l    move cursor right (space bar)

$    move to the last column on the current line
o    move cursor to the first column on current line
w    move cursor to the beginning of the next word
b    move cursor to the beginning of the previous word
H    move cursor to the top of the screen
M    move cursor to the middle of the screen
L    move cursor to the bottom of the screen

G    move to the last line in the file
nG   move to line n

r    replace character under cursor with next character typed
i    insert before cursor
a    append after cursor
A    append at end of line

x    delete   character under cursor
dd   delete line under cursor
dw   delete word under cursor

/    search for a word (forwards)
?    search a word (backwards)
:s   search and replace a word in current line
```

Start the **vi** text editor by typing **vi** followed by the name of the file to be created or modified. In this example, it is assumed that a new file called **myfile.txt** is to be created:

pi@raspberrypi: ~ $ **vi myfile.txt**

You should see the **vi** text editor screen displayed as in Figure 5.10. The name of the file being edited is displayed at the bottom of the screen.

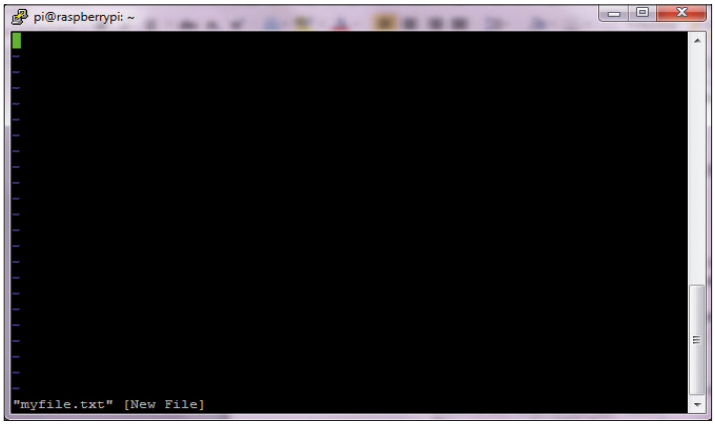

Figure 5.10 **vi** *text editor screen*

The **vi** editor is different from most other text editors in that it is not possible to start typing inside the editor window. The steps for editing this file are given below:

Step 1: The **vi** editor has different modes, and you must be in insert mode to be able to write to the window. Press *i* to enter insert mode. Then type in the following text (see Figure 5.11):

The vi text editor is a powerful text editor.
But it is not easy to use this editor.
This exercise should help you understand the basic commands.

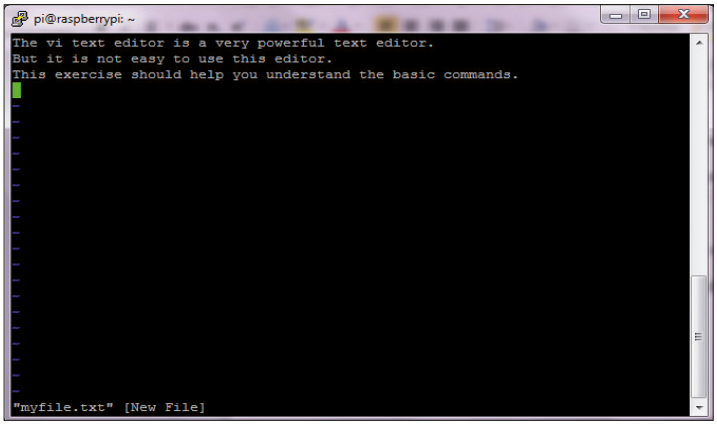

Figure 5.11 Entering the text

Step 2: To come out of the insert mode, press the ESC key. To save the file, type the characters **:w**. You can exit the editor after saving the changes by typing **:q**. Alternatively, you can type **ZZ** (note upper case) to save and exit. If you modified the file and attempt to quit without saving, you will get an error message. If you want to exit without saving the changes, simply type **:q!**

Step 3: Make sure you are in the command mode and type the character **/** followed by a word to search for this word in the text. For example, type **/editor** to search for the word **editor** (see Figure 5.12) in the text.

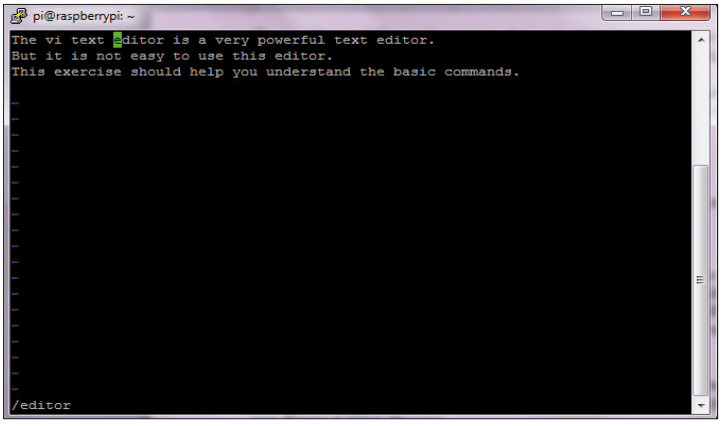

Figure 5.12 Searching for text

Step 4: Insert word **is** before word **editor**. Type **i** followed by **is** and space, and terminate insert mode by pressing the ESC key.

Step 5: Move cursor right by pressing the **l** key. Similarly, move the cursor left by pressing the **h** key. Move the cursor down (to the second line) by pressing the **j** key.

Step 6: Search for the word **this** and delete it. Type **/this** followed by the Enter key. Type **dw** to delete the word.

Step 7: Delete the second line where the cursor is on by typing **dd**

Step 8: Search for the word **help** and replace it with the word **guide**. Go to the line where the word help is. Type **/help**, then type **:s/help/guide/**

Step 9: You can search and replace a word in any other line than the current line. For this example, position the cursor on the first line. Change the word **basic** in the second line to **BASIC**. Type:

```
:1,2s/basic/BASIC/
```

Notice that you can specify the range of lines by separating them with a comma. In this example, the search starts from line 1 and terminates at line 2.

Chapter 6 • Creating and Running a Python Program

6.1 Overview

You will be programming your Raspberry Pi 5 using the Python programming language. It is worthwhile to look at the creation and running of a simple Python program on your Raspberry Pi 5 computer. In this chapter, the message **Hello From Raspberry Pi 5** will be displayed on your PC screen.

As described below, there are three methods that you can create and run Python programs on your Raspberry Pi 5.

6.2 Method 1 – Interactively from command prompt in console mode

In this method, you will log in to your Raspberry Pi 5 using the SSH and then create and run the Python program interactively. This method is excellent for small programs. The steps are as follows:

- Login to the Raspberry Pi 5 using SSH
- At the command prompt, enter **python**. You should see the Python command mode, which is identified by three characters "**>>>**"
- Type the program:

  print ("Hello From Raspberry Pi 5")

- The text will be displayed interactively on the screen as shown in Figure 6.1. Note that at the time of writing this book, the Python version was: 3.11.2.

```
pi@raspberrypi:~ $ python
Python 3.11.2 (main, Mar 13 2023, 12:18:29) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello from Raspberry Pi 5")
Hello from Raspberry Pi 5
>>>
```

Figure 6.1 Running a program interactively

- Type **Ctrl+z** to exit the program

6.3 Method 2 – Create a Python file in console mode

In this method, you will log in to your Raspberry Pi 5 using the SSH as before and then create a Python file. A Python file is simply a text file with the extension **.py**. You can use a text editor, e.g. the **nano** text editor, to create your file. In this example, a file called **hello.py** is created using the **nano** text editor. Figure 6.2 shows the contents of file **hello.py**. This figure also shows how to run the file under Python. Notice that the program is run by entering the command:

>>> python hello.py

```
pi@raspberrypi:~ $ ls
Bookshelf  Documents  hello.py  myfiles         Pictures  Templates  Videos
Desktop    Downloads  Music     mytestfile.txt  Public    test.txt
pi@raspberrypi:~ $ cat hello.py
print("Hello from Raspberry Pi 5")

pi@raspberrypi:~ $ python hello.py
Hello from Raspberry Pi 5
pi@raspberrypi:~ $
```

Figure 6.2 Creating and running a Python file

6.4 Method 3 – Create a Python file in Desktop GUI mode

In this method, you can log in to your Raspberry Pi 5 using either a directly connected terminal through the mini HDMI port, or if you don't have a monitor, you can log in to the Desktop using the VNC as described earlier, and then create and run your Python programs in GUI mode using the Thonny IDE. It is worthwhile at this stage to learn the basics of using the Thonny IDE.

The Thonny IDE

Start the Thonny IDE from the Desktop under the **Programming** menu. Figure 6.3 shows the Thonny startup menu.

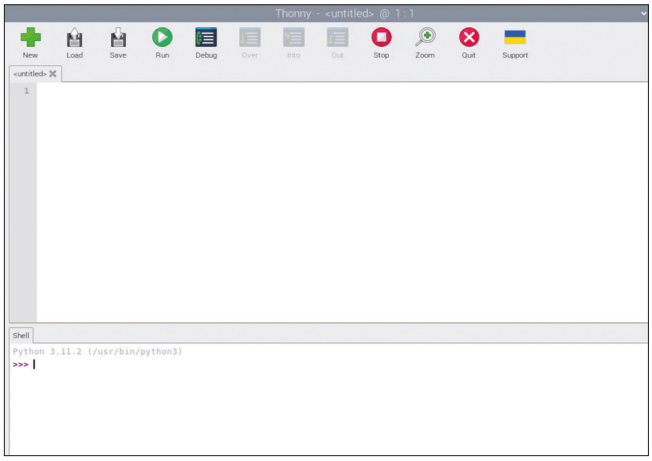

Figure 6.3 Thonny IDE startup menu

The screen consists of two parts: the upper part is where you write your programs. The lower part is the shell, where small interactive programs can be written. This part is mainly used for testing code snippets.

In the upper part contains the following menu items:

New: click to create a new program
Load: load an existing program from a folder on Raspberry Pi
Save: save an existing program on the screen to a file
Run: run the program on the screen
Debug: debug the program on the screen

Over: used by the debugger
Into: used by the debugger
Out: used by the debugger
Stop: stop a running program
Zoom: zoom the screen
Quit: Exit the Thonny IDE

The Thonny IDE must be configured before it is used to write and upload programs to your Raspberry Pi. Click the bottom-right corner of the screen to select your processor type and select **Local Python 3**. You are now ready to write your program. The steps are:

- Type the following code in the upper part of the screen:

```
print("Hello from Raspberry Pi 5")
```

- Click **File** → Save and save with the name hello.py (Figure 6.4)

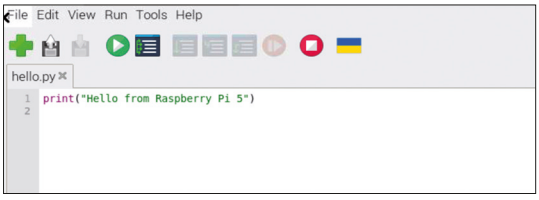

Figure 6.4 Type your program and save it

- Click the **Run** icon (green menu button at the top) to run the program. The output of the program will be displayed at the bottom of the screen as shown in Figure 6.5.

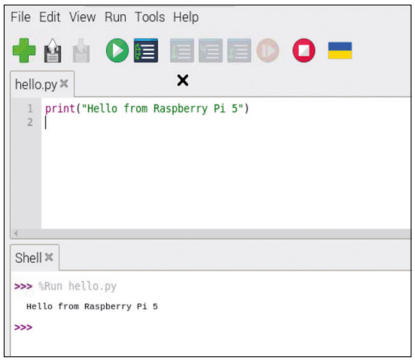

Figure 6.5 Run the program

You can run small programs in interactive mode by entering them at the lower part of the screen under **shell**. The results will be displayed under **shell** immediately.

6.5 Which method?

The choice of a method depends upon the size and complexity of a program. Small programs can be run interactively without creating a program file. Larger programs can be created as Python files, and then they can run either in the console mode or in Desktop GUI mode under the Thonny IDE. Running under the Thonny IDE has the advantage that justification of the code is corrected automatically as you write the code. In this book, the Thonny IDE is used for small programs and the nano editor is used for larger programs to create the program files.

Chapter 7 • Python Programming and Simple Programs

7.1 Overview

Python is an interpreted, interactive and object-oriented programming language. It was developed by Guido van Rossum in the 1980s at the National Institute for Mathematics and Computer Science in the Netherlands. It is derived from many other languages, including C, C++, Modula-3, SmallTalk, and Unix shells. The language is now maintained by a team of people at the Institute.

Python is interactive, which means that you can issue a command and see the result immediately without having to compile the command. It is interpreted, thus requiring no pre-compilation before it is run.

Python supports object-oriented techniques of programming. It is a beginners' language which is easy to learn and easy to maintain. Beginners can easily learn programming in a relatively short period of time. Python supports a large library of functions, which makes it powerful. The language is portable, meaning that it can run on several different popular platforms.

In this and next chapters, you will be learning the details of the Python programming language on the Raspberry Pi 5 computer, and see how you can write programs using this language. Many example programs are given to show how electronic engineers can use the Python language to help them in their calculations.

7.2 Variable names

Python variable names are case-sensitive and can start with a letter **A** to **Z** or **a** to **z** or an underscore character '_', followed by more letters or numbers 0 to 9. Some valid and invalid example variable names are given below:

SUM	-	valid
Sum	-	valid
SUm	-	valid
_total	-	valid
Cnt5	-	valid
8tot	-	invalid
%int	-	invalid
&xyz	-	invalid
My_Number	-	valid
@loop	-	invalid
_Account	-	valid

Note that variables **total**, **Total**, **TOTAL**, **ToTaL**, or **toTAL** are all different.

7.3 Reserved words

There are some words which are reserved for use by the Python interpreter and thus cannot be used as variable names by programmers. A list of these reserved words is given below. Notice that all the reserved words contain lower-case letters:

and	for	raise
assert	from	return
break	global	try
class	if	while
continue	import	with
def	in	yield
del	is	
elif	lambda	
else	not	
except	or	
exec	pass	
finally	print	

7.4 Comments

Comment lines in Python start with a hash sign **'#'**. All characters after the **#** sign are ignored by the Python interpreter. An example comment line is shown below:

```
# This is a comment line
```

Comments can also be inserted after a statement:

```
Sum = 0                    # Another comment
```

7.5 Line continuation

The line continuation character '\' can be used to continue a statement on the following lines. An example is shown below:

```
Sum =  a +\
       b +\
       c
```

Which is equivalent to:

```
Sum = a + b + c
```

7.6 Blank lines

A line containing no statements is ignored by the Python interpreter.

7.7 More than one statement on a line

It is permissible to have more than one statement on a single line by separating the statements with a semicolon character. An example is given below:

```
cnt = 5; sum = 0; tot = 20;
```

7.8 Indentation

In most programming languages, blocks of code are identified by using braces at the beginning and end of the block, or by identifying the end of the block using a suitable statement. e.g. END, WEND, or ENDIF. In the Python language, there are no braces or statements to indicate the start and end of a block. Instead, blocks of code are identified by line indentation. All statements within a block must be indented the same amount. The actual number of spaces used to indent a block is not relevant as long as all the statements in the block use the same number of spaces.

A valid block of code is given below (don't worry at this stage what the code does):

```
if j == 5:
        a = a + 1
        b = a + 2
else:
        a = 0
        b = 0
```

The following block of code is not valid since the indentation is not correct:

```
if j == 5:
        a = a + 1
        b = a + 2
else:
        a = 0
        b = 0
```

7.9 Python data types

Python supports the following data types:

- Numbers
- Strings
- Lists
- Dictionaries
- Tuples
- Sets
- Files

7.10 Numbers

Python supports the following numeric variable types:

- **int** - signed integer
- **long** - long integer
- **float** - floating-point real number

Complex number

Numbers can be represented in decimal, octal, binary, or hexadecimal. Long integers are shown with an upper-case letter **L**.

Some example numbers are shown below:

Integer

100	-	decimal
-67	-	decimal
500	-	decimal
0x20	-	hexadecimal
0b10000001	-	binary
0o2377	-	octal
202334567L	-	long decimal
0x3AEEFAE	-	hexadecimal

Floating point

2.355
23.780
-45.6
1.298
24.45E4

Complex

24.4+2,6j
0.78-4.2j
23.7j

We can assign numeric values to variables. These variable objects are created when values are assigned to them:

```
sum = 28
a = 0
```

We can delete a variable object by using the **del** statement:

```
del sum, a
```

We can assign a value to several variables at the same time:

```
w = x = y = z = 0
```

Similarly, we can have statements of the form:

```
w, x, y = 3, 5, 8
```

Which is equivalent to:

```
w = 3
x = 5
y = 8
```

we can perform the following mathematical operations on numbers:

Expression operators

+	addition
-	Subtraction
*	multiplication
/	division
>>	shift right
<<	shift left
**	power (exponentiation)
%	remainder

Bitwise operators

|	bitwise OR
&	bitwise AND
^	bitwise exclusive-or
~	bitwise complement

Some mathematical functions

pow(x,y)	same as x**y
abs(x)	absolute value of x
round(x,n)	round x to n digits from the decimal point
floor(x)	largest integer not greater than x
int(x)	convert x to integer
hex(x)	hexadecimal equivalent of integer x
bin(x)	binary equivalent of integer x
exp(x)	exponential of x
factorial(n)	factorial of number n
ceil(x)	smallest integer not less than x
log(x)	natural logarithm of x (base 2)
log10(x)	logarithm of x (base 10)

Some mathematical utility libraries

random	random number library
math	mathematics library

Figure 7.1 to Figure 7.3 show examples of using numbers in Python. Statement import is used to import a library to a Python program. The **math** library contains a large number of mathematical functions, such as logarithmic and trigonometric functions, square root, hyperbolic functions, angular conversion, and so on. Further details on these functions can be obtained from the following link:

https://docs.python.org/3/library/math.html

random library is useful to generate random numbers. The function **randint(a, b)** in this library generates an integer random number between integers **a** and **b** inclusive. Details of functions available in the random library can be obtained from the following link:

https://docs.python.org/2/library/random.html

```
>>> 28 + 35
63
>>> 22 * 6
132
>>> 2 ** 5
32
>>> 2 << 3
16
>>> 5 % 2
1
>>> abs(-100)
100
>>> 0x10
16
>>> 0o17
15
>>> 0b00001111
15
>>> (2 + 3j) * 3
(6+9j)
>>> hex(20)
'0x14'
>>> bin(15)
'0b1111'
>>>
```

Figure 7.1 Using numbers in Python

```
>>> int(23.256)
23
>>> float(4)
4.0
>>> 1/3.0
0.3333333333333333
>>> 10/4.0
2.5
>>> import math
>>> math.sqrt(16)
4.0
>>> math.pi
3.141592653589793
>>> math.floor(-3.5)
-4.0
>>> math.trunc(-4.5)
-4
>>> math.sin(30.0 * math.pi/180)
0.49999999999999994
>>> pow(2, 4)
16
>>> max(2,5,12,8)
12
>>> min(2,4,6,8)
2
```

Figure 7.2 Using numbers in Python

```
>>> a = 0b00001110
>>> bin(a & 0b11)
'0b10'
>>> bin(a | 0b11)
'0b1111'
>>> math.e
2.718281828459045
>>> math.floor(-2.7)
-3.0
>>> sum((1,2,3,4,5,6,7,8,9,10))
55
>>> import random
>>> random.randint(1, 5)
3
>>> random.randint(1, 5)
4
>>> random.randint(1, 5)
5
>>> (2 + 4j) + (4 + 3j)
(6+7j)
>>> (2.4 * 3), (5.0 / 2.0), math.sqrt(12.0)
(7.199999999999999, 2.5, 3.4641016151377544)
```

Figure 7.3 Using numbers in Python

7.11 Strings

In Python, strings are declared by enclosing characters between a pair of single or double quotation marks. An example is given below:

```
myname = "James Booth"
```

We can manipulate strings by extracting characters, joining two strings, assigning a string to another string, and so on. Some commonly used string manipulation operations are shown in Figure 7.4 and Figure 7.5.

```
>>> name = "John"
>>> surname = "Adams"
>>> full_name = name + surname
>>> print(full_name)
JohnAdams
>>> initial = name[0]
>>> print(initial)
J
>>> initials = name[0] + surname[0]
>>> print(initials)
JA
>>> print(name[0:3])
Joh
>>> print(name[:2])
Jo
>>> print(name[2:])
hn
>>> print(name*2)
JohnJohn
>>> print(name[0:2] + surname[2:4] + "end")
Joamend
>>> print(name + " " + surname)
John Adams
>>> print(len(name))
4
```

Figure 7.4 String manipulation operations

```
>>> name = "Smith"
>>> print(name[-1])
h
>>> print(name[-2])
t
>>> print(name.find('i'))
2
>>> name.replace('i', 'k')
'Smkth'
>>> numbers = "111,222,333,444,555"
>>> numbers.split(',')
['111', '222', '333', '444', '555']
>>> name
'Smith'
>>> name[0] = 's'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> name = 's' + name[1:]
>>> name
'smith'
```

Figure 7.5 String manipulation operations

Notice that a third index as the step can be used in string slicing operation. The step is added to the first offset until the second offset, and the character at this position is extracted. In the following example, the characters at positions 0, 2, 4, 6 are extracted:

```
>>> a = "computer"
>>> b = a[0:7:2]
>>> print(b)
cmue
```

7.11.1 String functions

Python supports many string functions. Some commonly used string functions are given below:

- capitalize() — change first letter of a string to upper case and all other characters to lower-case.
- count(str,beg,end) — find how many times **str** occurs in a string. String starting and ending positions should be specified
- find(str,beg,end) — determine if **str** occurs in a string. String starting and ending positions should be specified. The index is returned if the **str** is found, otherwise -1 is returned
- len(string) — return the length of a string
- isalpha() — return true if string contains all alphabetical characters
- isalnum() — return true if string contains alphabetical and numeric characters
- isdigit() — return true if string contains all digits
- islower() — return true if string contains all lower-case letters
- isupper() — return true if string contains all upper-case letters
- lower() — convert all upper-case characters to lower-case
- upper() — convert all lower case characters to upper case
- lstrip() — remove all leading white spaces
- rstrip() — remove all trailing spaces
- swapcase() — change case of all letters

Figure 7.6 shows examples of using some of the string functions.

```
>>> a = "computer"
>>> b = a.capitalize()
>>> print(b)
Computer
>>> print(a.count('p', 0, len(a)))
1
>>> print(a.find('p', 0, len(a)))
3
>>> print(len(a))
8
>>> b = a.upper()
>>> print(b)
COMPUTER
>>> a.isalpha()
True
>>> a.isdigit()
False
>>> b = a.swapcase()
>>> print(b)
COMPUTER
```

Figure 7.6 Using the string functions

7.11.2 Escape sequences

Escape sequences are special non-printable characters used to generate functions such as newline, tab, form feed, carriage return and so on. Escape sequences start with the character '****'. A list of the commonly used escape sequences is given below:

- \n newline
- \a bell
- \b backspace
- \f form feed
- \r carriage return
- \t horizontal tab
- \v vertical tab
- \xhh character defined by the 2-digit hexadecimal value hh

As an example, the following statement will display the letter 'a' followed by two newlines:

```
print("a\n\n")
```

7.12 Print statement

The print statement is one of the most commonly used statements. It displays text or numbers on the screen. Text is displayed by enclosing it in quotes. Numeric data is displayed by simply entering the variable name. The data to be displayed is enclosed in round brackets. Text and numeric data can be mixed in display outputs, and the type of the variable to be displayed can be declared using formatting characters. A list of the commonly used formatting characters is given below:

- %c character
- %s string
- %d signed integer
- %u unsigned integer

- %x lower case hexadecimal number
- %X upper-case hexadecimal number
- %f floating-point number
- %E exponential notation

Figure 7.7 shows some examples of using the print statement.

```
>>> first, last = 1, 100
>>> print("First = %d Last = %d" %(first, last))
First = 1 Last = 100
>>>
>>> name, age = "John", 21
>>> print("Name = %s Age = %d" %(name, age))
Name = John Age = 21
>>>
>>> a = 2.345
>>> print("a is %E" %(a))
a is 2.345000E+00
>>>
>>> a, b = 5, 10
>>> print("a is %d\n b is %d" %(a, b))
a is 5
 b is 10
>>>
>>> a = 100
>>> print("%X" %(a))
64
```

Figure 7.7 Using the print statement

7.13 List variables

List variables are variables separated by commas and enclosed in square brackets. The variables in a list can be of different types. The contents of a list can be accessed using square brackets to index the required item in the list. Indexing starts from 0. As with the strings, the '*' character can be used for repetition and the '+' character can be used for concatenation. Some examples are given below:

```
mylist = ['John', 'Adam', 230, 12.25, 'Peter', 89]
second = [30, 23]

s = mylist[0]           # s = 'John'
s = mylist[2]           # s = 230
s = mylist[2:4]         # s = 230, 12.25
s = mylist[3:]          # s = 12.25, 'Peter', 89
s = mylist * 2          # s = 'John', 'Adam', 230, 12.25, 'Peter', 89, 'John',
                        'Adam', 230, 12.25, 'Peter', 89
s = mylist + second     # s = 'John', 'Adam', 230, 12.25, 'Peter', 89, 30, 23
```

The contents of a list can be modified by assigning a new value to the required index position. For example, we can change the 2nd element of the list **mylist** from 230 to 100 as:

```
mylist[2] = 100
```

Python does not allow to reference items that are not present in a list. For example, the following statement gives an error message:

```
mylist[200]
```

Lists can be nested to form two-dimensional matrices. An example is given below:

```
M =  [[1, 2, 3],
      [4, 5, 6],
      [7, 8, 9]]
```

The nested list is indexed starting from [0][0]. For example, the elements of row 1 can be accessed as follows:

```
>>> M[1]                # Elements of row 1
[4, 5, 6]

>>> M[1][1]             # Element at row 1, column 1
5
```

The statement **L = [ ]** creates an empty list called **L**.

7.13.1 List functions

The Python language supports many list functions. Some commonly used list functions are given below:

- del([i:j]) delete elements from i to j-1
- list.append(x) append an item to the end of a list
- list.extend([x,y,z]) add several items to the list
- cmp(L1,L2) compare elements of lists L1 and L2
- len(L) length of list L
- max(L) item with the maximum value
- min(L) item with the minimum value
- list.count(x) returns how many times x occurs in a list
- list.index(x) returns the position of the first occurrence of x
- list.insert(i,x) inserts x at position i in the list
- list.remove(x) removes the indexed item from the list
- list.reverse() reverses a list
- list.sort() sorts a list
- list.pop() delete and return the last item

Figure 7.8 shows some examples of using the print statement.

```
>>> lst = ['A', 'B', 'C', 'D', 'E']
>>> del(lst[2])
>>> print(lst)
['A', 'B', 'D', 'E']
>>> del(lst[1:3])
>>> print(lst)
['A', 'E']
>>> lst.append('Z')
>>> print(lst)
['A', 'E', 'Z']
>>> lst.extend(['a', 'b', 'c'])
>>> print(lst)
['A', 'E', 'Z', 'a', 'b', 'c']
>>> print(lst.index('a'))
3
>>> print(lst.reverse())
None
>>> print(lst)
['c', 'b', 'a', 'Z', 'E', 'A']
>>> lst.sort()
>>> print(lst)
['A', 'E', 'Z', 'a', 'b', 'c']
>>> print(len(lst))
6
```

Figure 7.8 Using the list functions

7.14 Tuple variables

Tuples are similar to lists, but their contents cannot be changed, i.e. they are read-only. Also, tuple variables are enclosed in round brackets (parenthesis). Some examples are given below:

```
mytuple = ['John', 'Adam', 230, 12.25, 'Peter', 89]
second = [30, 23]

s = mytuple[0]          # s = 'John'
s = mytuple[2]          # s = 230
s = mytuple[2:4]        # s = 230, 12.25
s = mytuple[3:]         # s = 12.25, 'Peter', 89
s = mytuple * 2         # s = 'John', 'Adam', 230, 12.25, 'Peter', 89, 'John',
                        'Adam', 230, 12.25, 'Peter', 89
s = mytuple + second    # s = 'John', 'Adam', 230, 12.25, 'Peter', 89, 30, 23
```

The following statement is not valid, since we cannot change the contents of a tuple:

```
mytuple[2] = 200
```

7.15 Dictionary variables

Dictionaries are similar to hash tables with keys and values. Each key is separated from its value by a colon sign, the items are separated by command, and the whole thing is enclosed in curly brackets. The keys in a dictionary must have data types of numbers, strings, or tuples. The values can be of any data type. An example is given below:

```
mydict = {'Name': 'John', 'Surname': 'Adams', 'Age': 25}
s = mydict['Name']          # s = 'John'
s = mydict['Age']           # s = 25
s = mydict.keys()           # s = ['Age', 'Surname', 'Name']
s = mydict.values()         # s = [125, 'Adams', 'John']
```

7.15.1 Dictionary functions

The Python language supports a large number of dictionary functions. Some commonly used dictionary functions are given below:

- cmp(d1, d2) compare two dictionaries d1 and d2
- len() the number of items in a dictionary
- del(d[key]) delete an item from the dictionary
- d.clear remove all items from the dictionary
- d.keys() return a list of dictionary keys
- d.values() return a list of dictionary values

Figure 7.9 shows some examples of using the print statement.

```
>>> d = {'A':1, 'B':2, 'C':3, 'D':4}
>>> print(len(d))
4
>>> print(d.keys())
['A', 'C', 'B', 'D']
>>> print(d.values())
[1, 3, 2, 4]
>>> del(d['C'])
>>> print(d)
{'A': 1, 'B': 2, 'D': 4}
>>> d.clear()
>>> print(d)
{}
```

Figure 7.9 Using the dictionary functions

7.16 Keyboard input

Python provides the following function for reading data from the keyboard:

- input provides a prompted read. The data from the keyboard is returned as a string

Figure 7.10 shows examples of using the keyboard input function. Notice that the function returns a string. Therefore, if numeric data is entered, then it should be converted into a numeric data type before being used in mathematical operations.

```
>>> name = input("Enter your name: ")
Enter your name: John Smith
>>> a = input("Enter a number: ")
Enter a number: 5
>>> b = input("Enter another number: ")
Enter another number: 4
>>> c = int(a) + int(b)
>>> print(c)
9
>>> a = int(input("Enter a number: "))
Enter a number: 2
>>> b = int(input("Enter another number: "))
Enter another number: 4
>>> c = a * b
>>> print(c)
8
>>>
```

Figure 7.10 Keyboard input examples

7.17 Comparison operators

Valid Python comparison operators are:

- == checks if two operands are equal
- != checks if two operands are not equal
- > checks if the left operand is greater than the right one
- < checks if the left operand is less than the right one
- >= checks if the left operand is greater than or equal to the right one
- <= checks if the left operand is less than or equal to the right one

7.18 Logical operators

Valid Python logical operators are:

- and logical AND of the two operands
- or logical OR of the two operands
- not logical inverse of the operand

7.19 Assignment operators

- = assignment operator
- += compound add operator
- -= compound subtract operator
- *= compound multiply operator
- /= compound divide operator

7.20 Control of flow

In normal program flow, statements are executed sequentially one after another one. The flow control statements are used to make decisions and change the order of execution depending on the results of these decisions.

The Python programming language supports the following flow control statements:

- if
- if-else
- elif
- for
- while
- break
- continue
- pass

7.20.1 if, if...else, and elif

The general format of the if statement is:

```
        if expression: statement
or
        if expression:
                Statement 1
                Statement 2
        else:
                Statement 1
                Statement 2
```

Notice the use of indentation inside the **if** blocks and the colon character at the end of the **if** and **else** statements.

An example use of the if statement is:

```
if a == 5: print('a is 5')
```

If there is only one statement after the if, then it can be typed on the same line. If there is more than one statement then all the statements must be written on the next lines with the same amount of indentation. An example is given below:

```
if a == 100:
        x = 0
        y = 0
else:
        x = 1
        y = 10
```

The **elif** statement is used to check for different conditions in an **if** block. An example is given below:

```
if a > 10:
        b = 0
        c = 0
elif a == 10:
        b = 2
        c = 4
```

Notice that the if statements can be nested, as shown in the following example:

```
if a == 100:
        c = 0
        k = 1
if b == 10:
        c = 20
        m = 1
else:
        c = 23
```

7.20.2 for statement

The for statement is used to create loops (iteration) in programs. The general format of this statement is:

```
for variable in sequence:
        statements
```

Here, the sequence is evaluated first and the first item in the sequence is assigned to the variable and the statements are executed. Then the second item is assigned to the variable and the statements are executed. This continues until there are no more items in the sequence. An example use of the for statement is shown below:

```
for letter in "COMPUTER":
        print(letter)
```

The following will be displayed on the screen:

```
C
O
M
P
U
T
E
R
```

The **for** statement is commonly used to create loops in programs. The **range** statement denotes the range of the variable, as in the following example:

```
for cnt in range(0, 5):
        print(cnt)
```

The following will be displayed on the screen:

```
0
1
2
3
4
```

Notice that the upper value of the range is one less than the specified value. In the above example, range is from 0 to 4 and not to 5.

We can specify a step size in the last parameter when using the range statement, in the following example, the step size is 5 and the list takes values 0, 5, 10, 15, 20, 25:

```
List(range(0, 30, 5))
```

The **for** statement can be nested if desired.

7.20.3 while statement

The **while** statement can also be used to create loops (iteration) in programs. The general format of this statement is:

```
while expression:
        statements
```

The statements are executed while the expression evaluates to True. An example is given below:

```
cnt = 0
while cnt < 5:
        print(cnt)
        cnt = cnt + 1
```

The output of the program is as follows:

```
0
1
2
3
4
```

Notice that the statements that belong to the **while** statement must be indented. It is important to make sure that the expression is modified inside the loop; otherwise an infinite loop will be formed as shown in the following example:

```
cnt = 0
while cnt < 5:
        print(cnt)
```

7.20.4 continue statement

The **continue** statement is used in **for** and **while** loops, and this statement skips all the remaining statements in a loop and returns to the beginning of the loop. An example is given below. In this example, number 3 is not displayed by the print statement:

```
cnt = 0
while cnt < 5:
        cnt = cnt + 1
        if cnt == 3:
                continue
        print(cnt)
```

The output of this example is as follows:

1
2
4
5

7.20.5 break statement

The **break** statement is used in **for** and **while** loops, and this statement terminates the loop and execution continues with the next statement. An example is given below:

```
cnt = 0
while cnt < 5:
        cnt = cnt + 1
        if cnt == 3:
                break
        print(cnt)
```

The output of this program is as follows:

1
2

7.20.6 pass statement

The **pass** statement is used when a statement is required syntactically, but you do not want any command or code to execute. The **pass** statement is a null operation and nothing happens when it executes. An example is given below:

```
for letter in 'COMPUTER':
        if letter == 'P':
                pass
                print('Passed')
        print(letter)
```

The output of this program is:

C
O
M
Passed
P
U
T
E
R

We have covered the basic statements of the Python programming language. We will now develop example programs using the knowledge we have gained so far.

7.21 Example 1 – 4-Band resistor colour code identifier

In this example, the user enters the three colours of a 4-band resistor and the program calculates and displays the value of the resistor in ohms. The tolerance of the resistor is not displayed.

Background Information: Resistor values are identified by the following colour codes:

Black:	0
Brown:	1
Red:	2
Orange:	3
Yellow:	4
Green:	5
Blue:	6
Violet:	7
Grey:	8
White:	9

The first two colours determine the first two digits of the value, while the last colour determines the multiplier. For example, **red red red** corresponds to $22 \times 10^2 = 2200\ \Omega$.

Program Listing: Figure 7.11 shows the program listing (program: **resistor.py**). At the beginning of the program, a list called **colour** is created which stores the valid resistor colours. Then a heading is displayed, and a **while** loop is created which runs as long as string variable **yn** is equal to y. Inside the loop, the program reads the three colours from the keyboard using functions **input** and stores as strings in variables **FirstColour**, **SecondColour** and **ThirdColour**. These strings are then converted into lower case so that they are compatible with the values listed in the list box. The index values of these colours in the list are then found using function calls of the form **colours.index**. Remember that the index values start from 0. As an example, if the user entered red, then the corresponding index value will be 2. The resistor value is then calculated by multiplying the first colour number by 10 and adding to the second colour number. The result is then multiplied by the power of 10 of the third colour index. The final result is displayed on the screen. The program then asks whether the user wants to continue. If the answer is **y** then the program returns to the beginning; otherwise the program is terminated.

```
#================================================
#               RESISTOR COLOUR CODES
#               ---------------------
#
# The user enters the three colours of a resistor
# and the program calculates and displays the value
# of the resistor in Ohms
#
# Program: resistor.py
# Date   : October, 2023
```

```
# Author : Dogan Ibrahim
#================================================
colours = ['black','brown','red','orange','yellow','green',\
'blue','violet','grey','white']

print("RESISTOR VALUE CALCULATOR")
print("=========================")
yn = "y"

while yn == 'y':
  FirstColour = input("Enter First Colour: ")
  SecondColour = input("Enter Second Colour: ")
  ThirdColour = input("Enter Third Colour: ")
#
# Convert to lower case
#
  FirstColour = FirstColour.lower()
  SecondColour = SecondColour.lower()
  ThirdColour = ThirdColour.lower()
#
# Find the values of colours
#
  FirstValue = colours.index(FirstColour)
  SecondValue = colours.index(SecondColour)
  ThirdValue = colours.index(ThirdColour)
#
# Now calculate the value of the resistor
#
  Resistor = 10 * FirstValue  + SecondValue
  Resistor = Resistor * (10 ** ThirdValue)
  print("Resistance = %d Ohms" % (Resistor))
#
# Ask for more
#
  yn = input("\nDo you want to continue?: ")
  yn = yn.lower()
```

Figure 7.11 Program listing

The program was created using the **nano** text editor and then run from the command line by entering the following command:

pi@raspberrypi:~ $ **python resistor.py**

Figure 7.12 shows a typical run of the program.

```
pi@raspberrypi:~ $ python resistor.py
RESISTOR VALUE CALCULATOR
=========================
Enter First Colour: red
Enter Second Colour: red
Enter Third Colour: yellow
Resistance = 220000 Ohms

Do you want to continue?: n
pi@raspberrypi:~ $
```

Figure 7.12 Typical run of the program

7.22 Example 2 – Series or parallel resistors

This program calculates the total resistance of a number of series or parallel connected resistors. The user specifies whether the connection is in series or in parallel. Additionally, the number of resistors used is also specified at the beginning of the program.

Background Information: When a number of resistors are in series, then the resultant resistance is the sum of the resistance of each resistor. When the resistors are in parallel, then the reciprocal of the resultant resistance is equal to the sum of the reciprocal resistances of each resistor.

Program Listing: Figure 7.13 shows the program listing (program: **serpal.py**). At the beginning of the program, a heading is displayed, and the program enters into a **while** loop. Inside this loop, the user is prompted to enter the number of resistors in the circuit and whether they are connected in series or in parallel. The function **str** converts a number into its equivalent string. e.g. number 5 is converted into the string "5". If the connection is serial (mode equals to **'s'**) then the value of each resistor is accepted from the keyboard and the resultant is calculated and displayed on the screen. If on the other hand, the connection is parallel (mode is equal to **'p'**), then again the value of each resistor is accepted from the keyboard and the reciprocal of the number is added to the total. When all the resistor values are entered, the resultant resistance is displayed on the screen.

```
#================================================
#         RESISTORS IN SERIES OR PARALLEL
#         -------------------------------
#
# This program calculates the total resistance of
# serial or parallel connected resistors
#
# Program: serpal.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#================================================
print("RESISTORS IN SERIES OR PARALLEL")
print("==============================")
yn = "y"

while yn == 'y':
```

```
  N = int(input("\nHow many resistors are there?: "))
  mode = input("Are the resistors series (s) or parallel (p)?: ")
  mode = mode.lower()
#
# Read the resistor values and calculate the total
#
  resistor = 0.0

  if mode == 's':
    for n in range(0,N):
       s = "Enter resistor " + str(n+1) + " value in Ohms: "
       r = int(input(s))
       resistor = resistor + r
    print("Total resistance = %d Ohms" %(resistor))

  elif mode == 'p':
    for n in range(0,N):
      s = "Enter resistor " + str(n+1) + " value in Ohms: "
      r = float(input(s))
      resistor = resistor + 1 / r
    print("Total resistance = %.2f Ohms" %(1 / resistor))
#
# Check if the user wants to exit
#
  yn = input("\nDo you want to continue?: ")
  yn = yn.lower()
```

Figure 7.13 Program listing

Figure 7.14 shows a typical run of the program.

```
pi@raspberrypi:~ $ python serpal.py
RESISTORS IN SERIES OR PARALLEL
===============================

How many resistors are there?: 2
Are the resistors series (s) or parallel (p)?: s
Enter resistor 1 value in Ohms: 250
Enter resistor 2 value in Ohms: 100
Total resistance = 350 Ohms

Do you want to continue?: y

How many resistors are there?: 2
Are the resistors series (s) or parallel (p)?: p
Enter resistor 1 value in Ohms: 100
Enter resistor 2 value in Ohms: 100
Total resistance = 50.00 Ohms

Do you want to continue?: n
pi@raspberrypi:~ $
```

Figure 7.14 Typical run of the program

7.23 Example 3 – Resistive potential divider

Description: This case study calculates the resistances in a resistive potential divider circuit.

Background Information: Resistive potential divider circuits consist of two resistors. These circuits are used to lower a voltage to a desired value. Figure 7.15 shows a typical resistive potential divider circuit. Here, Vin and Vo are the input and output voltages respectively. R1 and R2 form the resistor pair used to lower the voltage from Vin to Vo. Many resistor pairs can be used to get the desired output voltage. Choosing large resistors draws little current from the circuit, and choosing small resistors draws larger current. In this design, the user specifies Vin, Vo, and R2. The program calculates the required R1 value to lower the voltage to the desired level. Additionally, the program displays the output voltage with the chosen physical resistors.

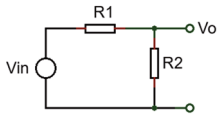

Figure 7.15 Resistive potential divider circuit

The output voltage is given by:

$$Vo = Vin\ R2 / (R1 + R2)$$

R1 is then given by:

$$R1 = (Vin - Vo)\ R2 / Vo$$

The above formula is used to calculate the required value of R1, given Vin, Vo, and R2

Program Listing: Figure 7.16 shows the program listing (program: **divider.py**). At the beginning of the program, a heading is displayed. The program then reads Vin, Vo, and R2 from the keyboard. The program calculates R1 and displays R1 and R2. The user is then asked to enter a chosen physical value for R1. With the chosen value of R1, the program displays Vin, Vo, R1, and R2 and asks the user whether the result is acceptable. If the answer to this question is **y** then the program terminates. If, on the other hand, the answer is **n** then the user is given the option of trying again.

```
#=======================================================
#             RESISTIVE POTENTIAL DIVIDER
#             ---------------------------
#
# This is a resistive potential divider circuit program.
# The program calculates the resistance values that will
# lower the input voltage to the desired value
#
```

```
# Program: divider.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#=======================================================
print("RESISTIVE POTENTIAL DIVIDER")
print("===========================")
R1flag = 1
R2flag = 0

while R1flag == 1:
  Vin = float(input("\nInput voltage (Volts): "))
  Vo = float(input("Desired output voltage (Volts): "))
  R2 = float(input("Enter R2 (in Ohms): "))
#
# Calculate R1
#
  R1 = R2 * (Vin - Vo) / Vo
  print("\nR1 = %3.2f Ohms R2 = %3.2f Ohms" %(R1, R2))
#
# Read chosen physical R1 and display actual Vo
#
  NewR1 = float(input("\nEnter chosen R1 (Ohms): "))

#
# Display and print the output voltage with chosen R1
#
  print("\nWith the chosen R1,the results are:")
  Vo = R2 * Vin / (NewR1 + R2)
  print("R1 = %3.2F R2 = %3.2f Vin = %3.2f Vo = %3.3f" %(NewR1,R2,Vin,Vo))
#
# Check if happy with the values ?
#
  happy = input("\nAre you happy with the values? ")
  happy = happy.lower()
  if happy == 'y':
    break
  else:
    mode = input("Do you want to try again? ")
    mode = mode.lower()
    if mode == 'y':
      R1flag = 1
    else:
      R1flag = 0
      break
```

Figure 7.16 Program listing

Figure 7.17 shows a typical run of the program.

```
pi@raspberrypi:~ $ python divider.py
RESISTIVE POTENTIAL DIVIDER
===========================

Input voltage (Volts): 10
Desired output voltage (Volts): 5
Enter R2 (in Ohms): 100

R1 = 100.00 Ohms R2 = 100.00 Ohms

Enter chosen R1 (Ohms): 100

With the chosen R1,the results are:
R1 = 100.00 R2 = 100.00 Vin = 10.00 Vo = 5.000

Are you happy with the values? y
pi@raspberrypi:~ $
```

Figure 7.17 Typical run of the program

7.24 Trigonometric functions

Python supports numerous trigonometric functions. The arguments to trigonometric functions must be in radians. The **math** library must be imported into the program before these functions can be used:

- sin(x) trigonometric sine
- cos(x) trigonometric cosine
- tan(x) trigonometric tangent
- asin(x) trigonometric arc sin
- atan(x) trigonometric arc tangent
- atan2(y, x) trigonometric atan(y/x)
- degrees(x) convert degrees into radians
- radians(x) convert radians into degrees

Some examples of using the trigonometric functions are given in Figure 7.18.

```
>>> import math
>>> print(math.radians(45))
0.785398163397
>>> print(math.degrees(1))
57.2957795131
>>> print(math.sin(math.radians(30)))
0.5
>>> print(math.cos(math.radians(60)))
0.5
>>> print(math.degrees(math.asin(0.5)))
30.0
```

Figure 7.18 Trigonometric function examples

7.25 User-defined functions

Functions are like small programs within a program. We can use functions to break up a complex program into several manageable sections, where each section can be implemented as a function. Functions enable us to reuse parts of our programs. For example, we can create a function to calculate the cube-root of a number and then call this function from different parts of our program. Another advantage of using functions is that they make it easier to maintain and update our programs.

A function that we create can be called from anywhere in a program. Functions have their own variables and their own commands. As we have seen in earlier parts of this chapter, Python has many built-in functions for various operations such as arithmetic, trigonometric, string manipulation and so on. User-defined functions are created by programmers. In this section, we shall be looking at how functions can be created and used in our programs.

A user-defined function consists of the following:

- Functions begin with the keyword **def**, followed by the function name, and round brackets, followed by a colon sign.
- Input arguments to the function must be placed inside the brackets at the beginning of the function definition.
- The body of a function must be indented with the same number of spaces on the left-hand side
- An optional text message can be displayed at the first line of a function to describe what the function does.
- A function must be terminated with the return statement

An example function, named **Mult** is given below. This function takes two numbers first and second as its arguments, multiplies them, and returns the result:

```
def Mult(first, second):
        "This is a simple multiplication function"
        result = first * second
        return result
```

A function is called from the main program by specifying the name of the function and enclosing any arguments in a pair of brackets. For example, to call the above function to multiply numbers 5 and 3 and sore the result in a variable called 'a', we include the following statement in our program:

```
a = Mult(5, 3)
```

We can also call a function by specifying the keyword arguments. i.e.:

```
a = Mult(first = 5, second = 3)
```

Figure 7.19 shows the above example in a Python program.

```
>>> def Mult(first, second):
...     "This is a simple multiplication function"
...     result = first * second
...     return result
...
>>> a = Mult(5, 3)
>>> print(a)
15
```

Figure 7.19 Creating and calling a function

Another example is shown in Figure 7.20. In this example, the function displays a string passed as an argument. Notice that there is no data returned from this function.

```
>>> def Prnt(strng):
...     print(strng)
...     return
...
>>> Prnt("Hello there")
Hello there
```

Figure 7.20 A function displaying a string

The variables used in a function are local to that function. Thus, for example, if there are two variables with the same name, one inside the function and the other one outside, changing the one inside the function does not change the one outside. Variables outside a function are called **global** variables, whereas the ones inside a function are called **local** variables. See Figure 7.21 for an example where the contents of variable **res** are not changed outside the function.

```
>>> def Mult(first, second):
...     res = first * second
...     return res
...
>>> res = 2
>>> a = Mult(3, 8)
>>> print(a)
24
```

Figure 7.21 Variables in a function are local

The rules for global variables are as follows:

- Global variables are variables assigned at the top of the program outside the function definitions
- Global names must be declared only if they are assigned within a function
- Global names may be referenced within a function without being declared

Therefore, by declaring a variable outside the functions and also inside a function but with the global keyword allows us to change its contents inside the function. An example is given below which identifies the use of global variables:

```
cnt = 10                        # variable cnt is global
def tstfunc():                  # function declaration
        global cnt              # variable cnt defined as global
        cnt = 200               # value of global cnt is changed

tstfunc()                       # function is called
print(cnt)                      # value of cnt is 200
```

As explained above, if the value of a global variable is not changed inside a function, then there is no need to define it as global. In the following code, there is no need to define **x** as global inside the function:

```
x = 10
y = 4

def tst():
        global y
        y = x + 2
```

It is important to note that the variables in a function call are passed **by value**. This means that the value of a parameter cannot be changed inside a function. An example is shown in Figure 7.22. In this example, notice that the value of variable **cnt** is not changed inside the function call.

```
>>> cnt = 2
>>> def Mult(first, second):
...     cnt = 5
...     return(first * second)
...
>>> a = Mult(5, 6)
>>> print(a)
30
>>> print(cnt)
2
```

Figure 7.22 Variables are passed by value

A function normally returns only one item to the calling program. In some applications, we may want to return more than one item to the calling program. This is easily done by returning a tuple and then unpacking it in the main program. An example is shown in Figure 7.23. In this example, the function **MyFunc** is declared with two arguments. The arguments are added and stored in a local variable called **sum**. Similarly, the difference of the arguments is stored in variable **diff**. The function returns both **sum** and **diff** as a tuple. The calling main program unpacks the returned data and stores in variables **x** and **y**.

```
>>> def MyFunc(a, b):
...     sum = a + b
...     diff = a - b
...     return sum, diff
...
>>> x, y = MyFunc(12, 5)
>>> print(x, y)
(17, 7)
```

Figure 7.23 Returning more than one variable from a function

7.26 Examples

Example 4

Write a program to read an angle from the keyboard in degrees and display the trigonometric sine of this angle. Repeat until the user stops the program.

Solution 4

The required program listing and example output are shown in Figure 7.24 (program: **trig.py**). The angle entered by the user is converted into floating point and is stored in variable angle. Then the trigonometric sine of this angle is displayed. The program continues until the user enters **n** in response to the prompt **Any more?**

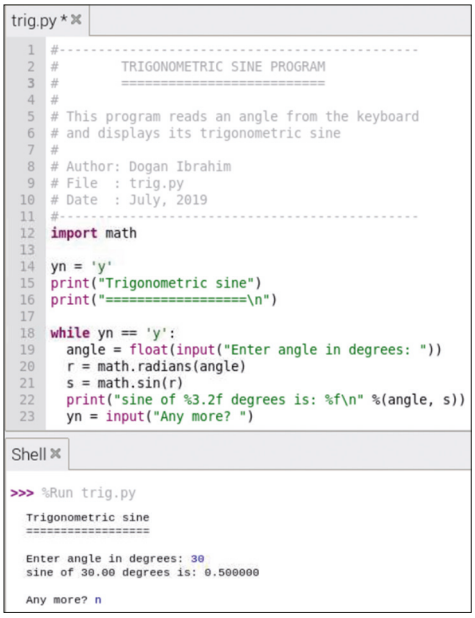

Figure 7.24 Program listing

This program was created and run using the Thonny IDE.

Example 5
Modify the program in Example 4 so that the user can choose between sine, cosine, and tangent.

Solution 5
The modified program listing and example output are shown in Figure 7.25 and Figure 7.26 (program: **trigall.py**). The user is given a menu with four choices: sine, cosine, tangent, exit. The angle is read from the keyboard and is converted into radians. The program then calculates the trigonometric value and displays on the screen. This process is repeated until the user selects the **exit** option.

```
#-------------------------------------------------
#    TRIGONOMETRIC SINE,COSINE,TANGENT PROGRAM
#    ========================================
#
# This program reads an angle from the keyboard
# and displays its trigonometric sine, cosine, or
# tangent depending on user choice. The angle is
# read in degrees,converted into radians and then
# the required trigonometric function is calculated
#
# Author: Dogan Ibrahim
```

```
# File  : trigall.py
# Date  : October, 2023
#-------------------------------------------------
import math

choice = '1'
while choice != '0':
  print(«Trigonometric Sine, Cosine, or Tangent»)
  print(«====================================\n»)
  print(«1. Sine»)
  print(«2. Cosine»)
  print(«3. Tangent»)
  print(«0. Exit»)
  choice = input(«Enter choice: «)

  if choice != '0':
    angle = float(input(«Enter angle in degrees: «))
    r = math.radians(angle)
    if choice == '1':
      s = math.sin(r)
      strng = «sine»
    elif choice == '2':
      s = math.cos(r)
      strng = «cosine»
    elif choice == '3':
      s = math.tan(r)
      strng = «tangent»
    print(strng + « of %3.2f degrees is: %f\n» %(angle, s))
print(«End of program»)
```

Figure 7.25 Modified program listing

```
pi@raspberrypi:~ $ python trigall.py
Trigonometric Sine, Cosine, or Tangent
====================================

1. Sine
2. Cosine
3. Tangent
0. Exit
Enter choice: 2
Enter angle in degrees: 30
cosine of 30.00 degrees is: 0.866025

Trigonometric Sine, Cosine, or Tangent
====================================

1. Sine
2. Cosine
3. Tangent
0. Exit
Enter choice: 0
End of program
pi@raspberrypi:~ $
```

Figure 7.26 Example output

This program was created using the **nano** text editor and then run using the command:

pi@raspberrypi:~ $ **python trigall.py**

Example 6
Write a program to tabulate the trigonometric sines of angles from 0º to 90º in steps of 5º.

Solution 6
The required program listing is shown in Figure 7.27 (program: **sinetable.py**). After displaying a heading, the for statement is used to create a loop. Variable angle takes values from 0 to 90 (inclusive) in steps of 5. The trigonometric sine is calculated and displayed.

```
#-------------------------------------------------
#      TRIGONOMETRIC SINE TABLE
#      ========================
#
# This program tabulates the trigonometric sine of
# angles from 0 to 90 degrees in steps of 5 degress
#
# Author: Dogan Ibrahim
# File  : sinetable.py
# Date  : October, 2023
#-------------------------------------------------
import math

print("TABLE OF TRIGONOMETRIC SINE")
print("==========================\n")
print("  ANGLE      SINE")

for angle in range(0, 95, 5):
  r = math.radians(angle)
  s = math.sin(r)
  print("  %d        %f" %(angle, s))

print("End of program")
```

Figure 7.27 Program listing

An example run of the program is shown in Figure 7.28.

```
pi@raspberrypi:~ $ python sinetable.py
TABLE OF TRIGONOMETRIC SINE
==========================

  ANGLE       SINE
  0         0.000000
  5         0.087156
  10         0.173648
  15         0.258819
  20         0.342020
  25         0.422618
  30         0.500000
  35         0.573576
  40         0.642788
  45         0.707107
  50         0.766044
  55         0.819152
  60         0.866025
  65         0.906308
  70         0.939693
  75         0.965926
  80         0.984808
  85         0.996195
  90         1.000000
End of program
pi@raspberrypi:~ $
```

Figure 7.28 Example run of the program

Example 7
Write a program to read metres from the keyboard. Convert into yards and inches and display the result.

Solution 7
The required program listing and example output are shown in Figure 7.29 program: **conv.py**). After displaying a heading, metres is read from the keyboard using the **input** statement. The value is then converted into yards and inches by multiplying with 1.0936 and 39.370 respectively. The results are displayed on the screen.

conv.py

```
#-----------------------------------------------
#                  CONVERSION PROGRAM
#                  ==================
#
# This program reads metres from the keyboard and
# converts and displays in yards and inches
#
# Author: Dogan Ibrahim
# File  : conv.py
# Date  : October, 2023
#-----------------------------------------------
print("Convert metres into yards and inches")
print("====================================")
metres = float(input("Enter metres: "))
yards = 1.0936 * metres
inches = 39.370 * metres
print("%f metres = %f yards, %f inches" %(metres, yards, inches))
print("End of program")
```

Shell

```
>>> %Run conv.py

  Convert metres into yards and inches
  ====================================
  Enter metres: 10
  10.000000 metres = 10.936000 yards, 393.700000 inches
  End of program

>>>
```

Figure 7.29 Program listing and example output

Example 8
Repeat Example 7 but do the conversion in a function called **Conv**. Show how this function can be called from the main program.

Solution 8

The required program listing and example output are shown in Figure 7.30 (program: **convfunc.py**). Function Conv is declared at the beginning of the program. Metres to be converted into yards and inches is passed as an argument to the function. The function returns the yards and inches in a tuple. The main program reads the metres from the keyboard and calls the function Conv. The result is displayed on the screen.

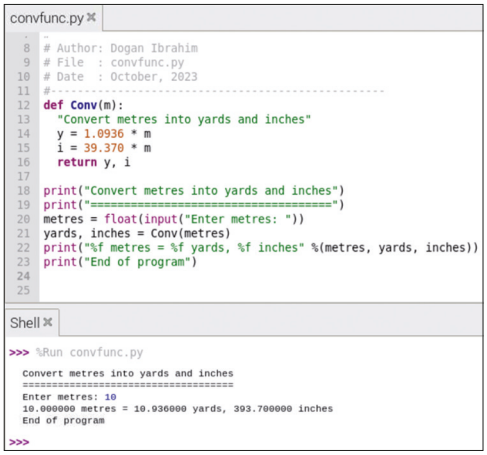

Figure 7.30 Program listing and example output

Example 9

Write a function called **Cyl** to calculate the area and volume of a cylinder, given its radius and height. Use this function in a main program.

Solution 9

The area and volume of a cylinder are given by the formula:

$$\text{Area} = 2\pi rh$$
$$\text{Volume} = \pi r^2 h$$

The required program listing and example output are shown in Figure 7.31 (program: **cylinder.py**). The radius and height of the cylinder are passed as arguments to a function which calculates the area and volume of the cylinder and returns the results to the main program, which are displayed on the screen.

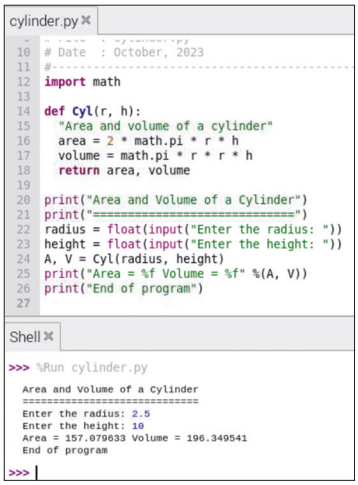

Figure 7.31 Program listing and example output

Example 10

Write a calculator program to carry out the four simple mathematical operations of addition, subtraction, multiplication, and division on two numbers received from the keyboard.

Solution 10

The required program listing is shown in Figure 7.32 (program: **calc.py**). Two numbers are received from the keyboard and stored in variables **n1** and **n2**. Then, the required mathematical operation is received and it is performed. The result, stored in the variable **result**, is displayed on the screen. The user is given the option of terminating the program.

```
#-----------------------------------------------
#              CALCULATOR PROGRAM
#              ==================
#
# This is a simple calculator program that can
# carry out 4 basic arithmetic opertions
#
# Author: Dogan Ibrahim
# File  : calc.py
# Date  : October, 2023
#-----------------------------------------------
any = 'y'
while any == 'y':
   print("\nCalculator Program")
   print("==================")

   n1 = float(input("Enter first number: "))
   n2 = float(input("Enter second number: "))
   op = input("Enter operation (+-*/): ")
```

```
if op =="+":
   result = n1 + n2
elif op == "-":
   result = n1 - n2
elif op == "*":
   result = n1 * n2
elif op == "/":
   result = n1 / n2
print("Result = %f" %(result))
any = input("\nAny more (yn): ")
```

Figure 7.32 Program listing

An example run of the program is shown in Figure 7.33.

```
pi@raspberrypi:~ $ python calc.py

Calculator Program
==================
Enter first number: 25
Enter second number: 3
Enter operation (+-*/): *
Result = 75.000000

Any more (yn): n
pi@raspberrypi:~ $
```

Figure 7.33 Example output

Example 11
Write a program to simulate double dice. i.e. to display two random numbers between 1 and 6 every time it is run.

Solution 11
The required program listing and example output are shown in Figure 7.34 (program: **dice.py**). Here, the random number generator **randint** is used to generate random numbers between 1 and 6 when the Enter key is pressed. The program is terminated when the letter **X** is entered.

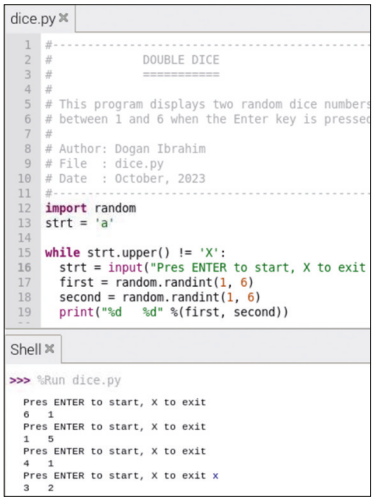

Figure 7.34 Program listing and example output

Example 12

Write a program to use functions to calculate and display the areas of shapes: square, rectangle, triangle, circle, and cylinder. The sizes of the required sides should be received from the keyboard.

Solution 12

The areas of the shapes to be used in the program are as follows:

Square: side = a	area = a2
Rectangle: sides a, b	area = ab
Circle: radius r	area = πr2
Triangle: base b, height h	area = bh/2
Cylinder: radius r, height h	area = 2πrh

The required program listing is shown in Figure 7.35 (program: **areas.py**). A different function is used for each shape, and the sizes of the sides are received inside the functions. The main program displays the calculated area for the chosen shape.

```
#----------------------------------------------
#              AREAS OF SHAPES
#              ===============
#
# This program calculates and displays the areas
# of various geometrical shapes
# of numbers in a list
#
# Author: Dogan Ibrahim
# File  : areas.py
```

```
# Date  : October, 2023
#-------------------------------------------
import math

def Square(a):                          # square
  return a * a

def Rectangle(a, b):                    # rectangle
  return(a * b)

def Triangle(b, h):                     # triangle
  return(b * h / 2)

def Circle(r):                          # circle
  return(math.pi * r * r)

def Cylinder(r, h):                     # cylinder
  return(2 * math.pi * r * h)

print("AREAS OF SHAPES")
print("===============\n")
print("What is the shape?: ")

shape = input("Square (s)\nRectangle(r)\nCircle(c)\n\
Triangle(t)\nCylinder(y): ")

shape = shape.lower()
if shape == 's':
  a = float(input("Enter a side of the square: "))
  area = Square(a)
  s = "Square"
elif shape == 'r':
  a = float(input("Enter one side of the rectangle: "))
  b = float(input("Enter other side of the rectangle: "))
  area = Rectangle(a, b)
  s = "Rectangle"
elif shape == 'c':
  radius = float(input("Enter radius of the circle: "))
  area = Circle(radius)
  s = "Circle"
elif shape == 't':
  base = float(input("Enter base of the triangle: "))
  height = float(input("Enter height of the triangle: "))
  area = Triangle(base, height)
  s = "Triangle"
elif shape == 'y':
```

```
    radius = float(input("Enter radius of cylinder: "))
    height = float(input("Enter height of cylinder: "))
    area = Cylinder(radius, height)
    s = "Cylinder"

print("Area of %s is %f" %(s, area))
```

Figure 7.35 Program listing

An example run of the program is shown in Figure 7.36.

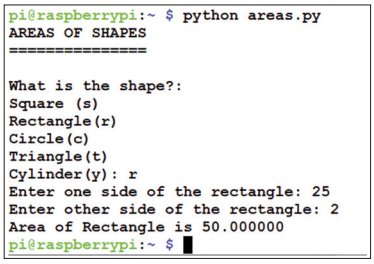

Figure 7.36 Example output

7.27 Recursive functions

Recursive functions are functions that call themselves either directly or indirectly, and such functions are supported by Python. Although the topic of recursive functions is an advanced topic, an example is given in Figure 7.37 to illustrate the principles of such functions. This recursive function implements the factorial operation. Detailed analysis of recursive functions is beyond the scope of this book.

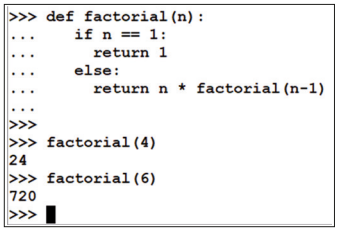

Figure 7.37 Recursive factorial function

7.28 Exceptions

There may be major errors in our programs, such as dividing by zero, file privilege error, and so on. Normally, when Python encounters such errors, it cannot handle them and the program crashes.

One way to handle such errors orderly and avoid crashes is to use exception handling in our programs. The basic method is that whenever an error occurs, the program detects this error and takes appropriate measures to handle the error and continue to execute normally. Exception handling is also useful if we wish to terminate a running program in an

orderly manner, for example to shut down any input-output operations when the program is terminated asynchronously by the user (e.g. by pressing the Ctrl+C key).

The statements try and except are used to handle unexpected errors or terminations in our programs. The general format of exception handling is as follows:

```
try:
        Normal program statements
        Normal program statements
except condition 1:
        if condition 1 type error occurs, then execute this block of code
                ……..
                ……..
except condition 2:
        if condition 2 type error occurs, then execute this block of code
                ……..
                ……..
else:
        if there are no errors detected, then execute this block of code
                ……
                ……
```

We can use the except statement with no condition to handle any type of exception. Some of the commonly used exceptions are:

exception EOFError:	end-of-file condition is reached while reading data
exception ImportError:	import statement could not load a module
exception IndexError:	sequence subscript is out of range
exception KeyError:	a dictionary key is not found in the set of existing keys
exception KeyboardInterrupt:	user hit the interrupt key (normally the Ctrl+C or Delete key)
exception MemoryError:	operation ran out of memory
exception OverFlowError:	arithmetic operation resulted in overflow
exception RuntimeError:	an error is detected that does not fall in any other categories
exception ValueError:	an operation or function receives an argument that has the right type but an inappropriate value
exception ZeroDivisionError:	a division by zero occurred

Some examples of using exceptions in programs are given below.

Example 13
Write a program to wait for an input from the keyboard. Terminate the program orderly when the Ctrl+C keys are pressed on the keyboard.

Solution 13

Figure 7.38 shows the program listing (program: except1.py). Exception KeyboardInterrupt is used in this program. The message End of Program is displayed when Ctrl+C key combination is pressed on the keyboard.

```
#=====================================================
#      KeyboardInteerupt EXCEPTION
#
# This program detects the keyboard entry Cntrl+C and
# the program is teminated orderly after the message
# End of Program is displayed
#
# Author : Dogan Ibrahim
# File   : except1.py
# Date   : October, 2023
#=====================================================
try:
   mode = input("Enter Cntrl+C to terminate the program: ")
except KeyboardInterrupt:
   print("\nEnd of Program")
```

Figure 7.38 Program listing

Example 14

Write a program to detect division by zero and to display the message Divide by Zero when this exception is detected.

Solution 14

Figure 7.39 shows the program listing (program: except2.py). Here, the program is forced to divide a number by zero and this is detected as an exception and the program displays a message when this occurs.

```
#=====================================================
#      ZeroDivisionError EXCEPTION
#
# This program detects when a number is divided by zero
# and generates an exception to display a message
##
# Author : Dogan Ibrahim
# File   : except2.py
# Date   : October, 2023
#=====================================================
print("Divide by zero exception")

try:
   s = 10 / 0
```

```
except  ZeroDivisionError:
   print("Divide by Zero")
```

Figure 7.39 Program listing

When the program is run, it displays the following message:

```
Divide by zero exception
Divide by Zero
```

7.29 try/final exceptions

Statement finally can be used in exception handling. Try/finally combination specifies exception, where the block beginning with finally is always executed on the way out, regardless of whether an exception occurs in the try block. An example is given below:

Example 15

Write a program to look for KeyboardInterrupt exception and display the message "Exception not occurred" if an exception has not occurred.

Solution 15

Figure 7.40 shows the program listing (program: except3.py). The block inside finally is executed regardless of whether an exception occurs.

```
#=======================================================
#          try/finally In EXCEPTION
#
# This program detects the keyboard entry Cntrl+C and
# displays the message Keyboard Interrupt if interrupt
# occurs. Message Continue is displayed regardless of
# whether an exception occurrede
#
# Author : Dogan Ibrahim
# File   : except3.py
# Date   : October, 2023
#=======================================================
try:
   mode = input("Enter Cntrl+C to terminate the program: ")
except KeyboardInterrupt:
   print("\nKeyboard Interrupt")
finally:
   print("\nContinue")
```

Figure 7.40 Program listing

When the program is run, the following is displayed

Enter Ctrl+C to terminate the program:
After entering Ctrl+C:
Keyboard Interrupt

7.30 Date and time

In some applications it may be necessary to get the current date and time. Python supports a number of functions to get the current date and time. Module time must be imported before these functions can be used. Some of the commonly used date and time functions are as follows:

- time.localtime() returns the current date and time in the following format:
  time.struct_time(tm_year=2013,tm_mon=12,tm_mday=18, tm_hour=12,tm_min=45,tm_sec=3,tw_wday=2,tm_yday=352, tm_isdst=0)
- time.asctime() returns the date and time in standard readable format
- time.clock() returns the current CPU time in seconds
- time.ctime() returns the current date and time
- time.time() returns the current time in seconds since the epoch
- time.sleep(x) suspends the calling program for x seconds

Some examples are given in Figure 7.41.

```
>>> import time
>>> print(time.localtime())
time.struct_time(tm_year=2023, tm_mon=10, tm_mday=6, tm_hour=14, tm_min=31,
ec=0, tm_wday=4, tm_yday=279, tm_isdst=1)
>>>
>>> print(time.asctime())
Fri Oct  6 14:31:05 2023
>>>
>>> print(time.ctime())
Fri Oct  6 14:31:15 2023
>>>
>>> print(time.time())
1696599083.4537978
>>>
```

Figure 7.41 Example date and time functions

The datetime module can also be used for date and time functions. This module must be imported to use these functions. Some examples of date functions are shown in Figure 7.42.

```
>>> from datetime import date
>>> print(date.today())
2023-10-06
>>>
>>> print(date.today().year)
2023
>>>
>>> print(date.today().month)
10
>>>
>>> print(date.today().day)
6
>>>
```

Figure 7.42 Examples of using the datetime date functions

The function strftime(format) is very useful as it can be used to format a date and time string. Some examples of using this function are given in Figure 7.43.

```
>>> from datetime import datetime
>>> print(datetime.now().strftime("%Y:%m"))
2023:10
>>>
>>> print(datetime.now().strftime("%H:%M:%S"))
14:41:08
>>>
>>> print(datetime.now().strftime("%d:%m:%Y"))
06:10:2023
>>>
>>> print(datetime.now().strftime("%d:%m:%Y_%H:%M:%S"))
06:10:2023_14:42:07
>>>
```

Figure 7.43 Examples of using strftime

7.31 Creating your own modules

In some applications, we may want to create our own Python modules and import them into our programs. Python modules are simply .py program files. Writing a module is just like writing any other Python program. Modules can contain functions, classes, and variables.

A simple module called msg.py is shown below:

```
def hello():
   print("Hello there!")
```

We can now import this module into our Python programs. An example program called myprog.py is shown below:

```
import msg
msg.hello()
```

Running the program: python myprog.py will display the following output:

```
Hello there!
```

We can also modify our program myprog.py and import and then call the module as follows:

```
from msg import hello
hello()
```

We can use variables in our module as shown below:

msg.py

```
def hello():
   print("Hello there!")

name = "Jones"
```

myprog.py

```
import msg
msg.hello()
print(msg.name)
```

The program will display:

```
Hello there!
Jones
```

Aliases can be created for modules. This is shown in the following code:

myprog.py

```
import msg as tst
tst.hello()
```

Will display the output:

```
Hello there!
```

An example module is given below that calculates the cube of a number

Example 16

Write a module that calculates the cube of the integer number passed to it. Show how this module can be imported and used in a program.

Solution 16

Figure 7.44 shows the module listing (program: cubeno.py). The function cube inside cubeno.py has the number as its argument. The cube of this number is calculated and returned. Figure 7.45 shows the program (program: myprog.py). As an example, when the number is 3, the output from the program is:

```
       Cube of 3 is: 27

def cube(N):
  r = N * N * N
  return r
```

Figure 7.44 Program cubeno.py listing

```
import cubeno
n= 3
res = cubeno.cube(n)
print("Cube of %d is: %d" %(n,res))
```

Figure 7.45 Program myprog.py

Module Search Path: When a module is to be imported, Python looks at the following folders in the order given:

The folder from which the module is called (where the calling main program is)

- The list of directories contained in the PYTHONPATH environment variable.
- Installation dependent list of directories configured when Python was installed

The Python search path can be displayed by entering the following command interactively:

```
>>> import sys
>>> sys.path
```

The display on the author's computer is shown in Figure 7.46.

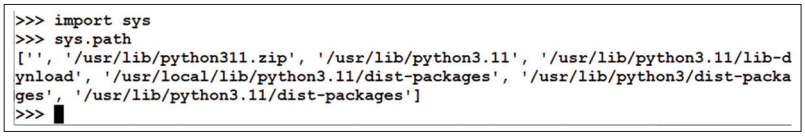

Figure 7.46 Python path display

To make sure that your module is found by Python, you can do one of the following:

- Put the module program file in the folder where your main program is
- Modify PYTHONPATH environment variable to contain the folder where the module program is
- Put the module program in one of the folders already contained in the PYTHONPATH

Chapter 8 • Raspberry Pi 5 LED Projects

8.1 Overview

This chapter is about the Raspberry Pi 5 hardware interface and using LEDs in simple projects. The Raspberry Pi 5 is connected to external electronic circuits and devices using its GPIO (General Purpose Input Output) port connector. This is a 2.54 mm, 40-pin expansion header, arranged in a 2 × 20 strip as shown in Figure 8.1. The I/O ports are numbered as GPIO nn

PIN	NAME	NAME	PIN
01	3.3V DC Power	5V DC Power	02
03	GPIO02 (SDA1,I²C)	5V DC Power	04
05	GPIO03 (SDL1,I²C)	Ground	06
07	GPIO04 (GPCLK0)	GPIO14 (TXD0, UART)	08
09	Ground	GPIO15 (RXD0, UART)	10
11	GPIO17	GPIO18(PWM0)	12
13	GPIO27	Ground	14
15	GPIO22	GPIO23	16
17	3.3V DC Power	GPIO24	18
19	GPIO10 (SP10_MOSI)	Ground	20
21	GPIO09 (SP10_MISO)	GPIO25	22
23	GPIO11 (SP10_CLK)	GPIO08 (SPI0_CE0_N)	24
25	Ground	GPIO07 (SPI0_CE1_N)	26
27	GPIO00 (SDA0, I²C)	GPIO01 (SCL0, I²C)	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12 (PWM0)	32
33	GPIO13 (PWM1)	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

Figure 8.1 Raspberry Pi 5 GPIO pins (https://linuxhint.com/gpio-pinout-raspberry-pi)

8.2 Raspberry Pi 5 GPIO pin definitions

When the GPIO connector is at the far side of the board, the pins starting from the left of the connector are numbered as 1, 3, 5, 7, and so on, while the ones at the top are numbered as 2, 4, 6, 8 and so on (Figure 8.2)

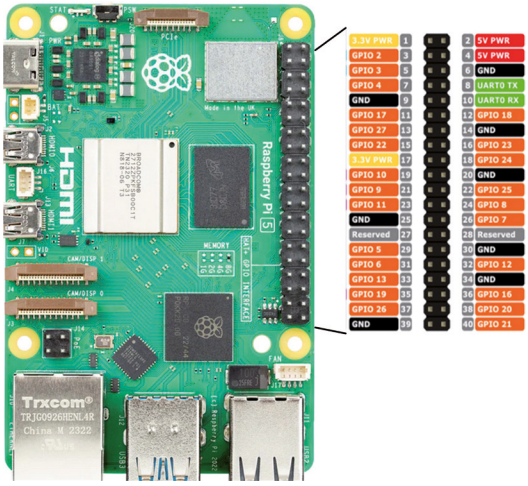

Figure 8.2 GPIO pin numbering

The GPIO provides 26 general-purpose bidirectional I/O pins. Some of the pins have multiple functions. For example, pins 3 and 5 are the GPIO2 and GPIO3 input-output pins respectively. These pins can also be used as the I²C bus SDA and SCL pins respectively. Similarly, pins 9, 10, 11 and 19 can either be used as general-purpose input-output pins, or as the SPI bus pins. Pins 8 and 10 are reserved for UART serial communication.

Two power outputs are provided: +3.3 V and +5.0 V. The GPIO pins operate at +3.3 V logic levels (not like many other computer circuits that operate with +5 V). A pin can either be an input or an output. When configured as an output, the pin voltage is either 0 V (logic 0) or +3.3 V (logic 1). Raspberry Pi 5 is normally operated using an external power supply (e.g. a mains adapter) with +5 V output. A 3.3 V output pin can supply up to 16 mA of current. The total current drawn from all output pins should not exceed the 51 mA limit. Care should be taken when connecting external devices to the GPIO pins, as drawing excessive currents or short-circuiting a pin can easily damage your Raspberry Pi. The amount of current that can be supplied by the 5 V pin depends on many factors, such as the current required by the Pi itself, current taken by the USB peripherals, camera current, micro-HDMI port current, and so on.

When configured as an input, a voltage above +1.7 V will be taken as logic 1, and a voltage below +1.7 V will be taken as logic 0. Care should be taken not to supply voltages greater than +3.3 V to any I/O pin, as large voltages can easily damage your Raspberry Pi. The Raspberry Pi 5, like others in the family, has no overvoltage protection circuitry.

8.3 Project 1 – Flashing an LED

Description: This is perhaps the easiest hardware project you can design using your Raspberry Pi 5. In this project, you will connect an LED to one of the ports of the Raspberry Pi 5 and then flash the LED once a second. The aim of this project is to show how a simple Python program can be written and then run from a file. The project also shows how to connect an LED to a Raspberry Pi 5 GPIO pin. In addition, the project shows how to use the GPIO library to configure and set a GPIO pin to logic 0 or 1.

Block diagram: The block diagram of the project is shown in Figure 8.3

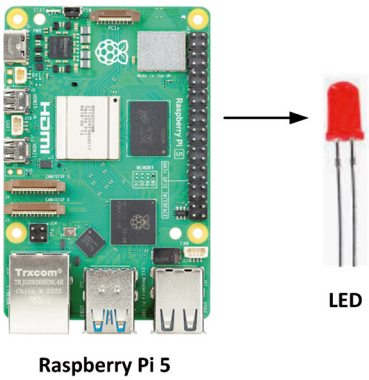

Figure 8.3 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 8.4. A small LED is connected to port pin GPIO 17 (pin 11) of the Raspberry Pi 5 through a current limiting resistor. The value of the current limiting resistor is calculated as follows:

The output high voltage of a GPIO pin is 3.3 V. The voltage across an LED is approximately 1.8 V. The current through the LED depends upon the type of LED used and the amount of required brightness. Assuming that we are using a small LED, we can assume a forward LED current of about 3 mA. Then, the value of the current limiting resistor is:

$$R = (3.3 - 1.8) / 0.003 = 500\ \Omega.$$ We can choose a 470 Ω resistor.

In Figure 8.4 the LED is operated in current sourcing mode where a high output from the GPIO pin drives the LED. The LED can also be operated in current sinking mode, where the other end of the LED is connected to +3.3 V supply and not to the ground. In current sinking mode, the LED is turned ON when the GPIO pin is at logic low.

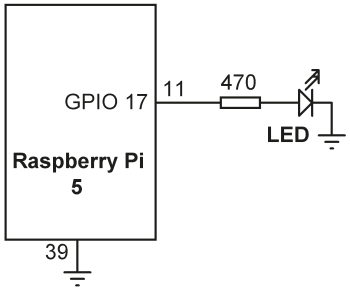

Figure 8.4 Circuit diagram of the project

Construction: The project is constructed on a breadboard as shown in Figure 8.5. Jumper wires are used to connect the LED to the GPIO port. Notice that the short side of the LED must be connected to ground.

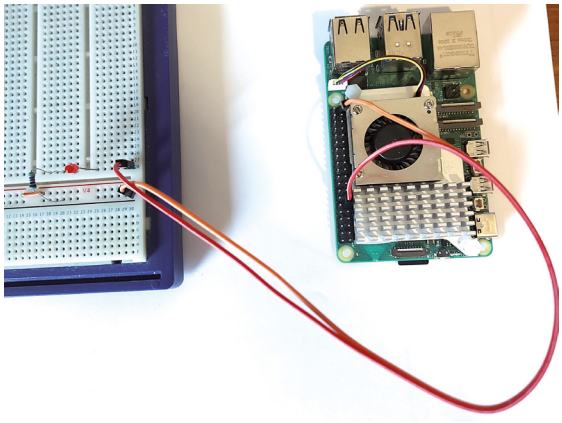

Figure 8.5 Constructing the project on a breadboard

Program listing: The program is called **LED.py** and the listing is shown in Figure 8.6. The program was written using the **nano** text editor. At the beginning of the program, the **gpiozero** and the **time** modules are imported to the project. The rest of the program is executed indefinitely in a **while** loop, where the LED is turned on and off with a one-second delay between each output change. Press Ctrl+C to terminate the program.

```
#-------------------------------------------------------
#
#                      FLASHING LED
#                      ============
#
# In thisproject a small LED is connected to GPIO 17 of
# the Raspberry Pi 5. The program flashes the LED every
# second.
#
# Program: LED.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#-------------------------------------------------------
from gpiozero import LED       # import gpiozero
from time import sleep         # import time library

led = LED(17)

while True:
  led.on()                     # turn ON LED
  sleep(1)                     # wait 1 second
  led.off()                    # turn OFF LED
  sleep(1)                     # wait 1 second
```

Figure 8.6 Program listing of the project

The program is run from the console mode as follows:

pi@raspberrypi ~ $ **python LED.py**

If you wish to run the program from the GUI Desktop environment, you should use the **VNC Viewer** to get into the GUI desktop screen (unless you have a monitor connected to the Raspberry Pi 5 via a micro-HDMI cable). Then, click the **Applications menu** → **Programming** → **Thonny**.

Click **File** and open file **LED.py**, or type in the program if it is not already in your default directory. Now, click **Run** to run the program. You should see the LED flashing every second. To terminate the program, close the screen by clicking the **STOP** button.

Note: You can copy the programs from your Raspberry Pi 5 home directory to your PC using the **winSCP** file copy program (available free of charge on the Internet).

8.4 Project 2 – Alternately flashing LEDs

Description: This project is similar to the previous one, but here two LEDs are used, and they flash alternately every second. The aim of this project is to show how more than one LED can be connected to the Raspberry Pi 5.

Block diagram: The block diagram of the project is shown in Figure 8.7

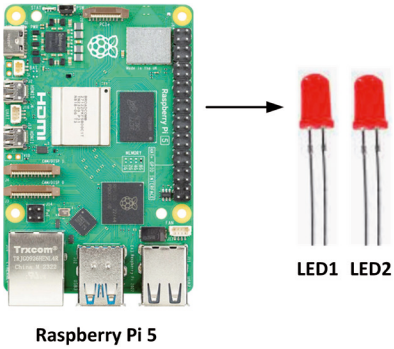

Figure 8.7 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 8.8. Two small LEDs are connected to port pins GPIO 17 (pin 11) and GPIO 27 (pin 13) of the Raspberry Pi 5 through current limiting resistors.

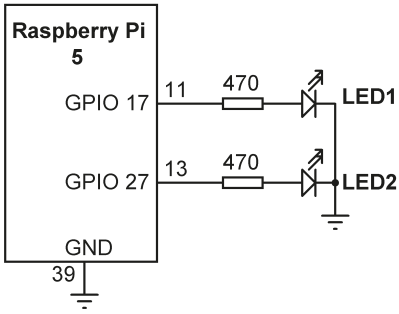

Figure 8.8 Circuit diagram of the project

Program listing: The program is called **alternate.py** and the listing is shown in Figure 8.9. The program was written using the **nano** text editor. At the beginning of the program, the **gpiozero** and the **time** modules are imported to the project. The rest of the program is executed indefinitely in a **while** loop where the LEDs are turned on and off alternately with one-second delay between each output. Press Ctrl+C to terminate the program.

```
#------------------------------------------------------
#
#               ALTERNATELY FLASHING LEDS
#               =========================
#
# In thisproject two small LEDs is connected to GPIO 17 and
# GPIO 27 of the Raspberry Pi 5. The program flashes the LEDs
# alternately every second
#
# Program: alternate.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#------------------------------------------------------
from gpiozero import LED
from time import sleep          # import time library

led1 = LED(17)                  # LED1 at GPIO 17
led2 = LED(27)                  # LED2 at GPIO 27

while True:
  led1.on()                     # Turn ON LED1
  led2.off()                    # Turn OFF LED2
  sleep(1)                      # Wait 1 second
  led1.off()                    # Turn OFF LED1
  led2.on()                     # Turn ON LED2
  sleep(1)                      # Wait 1 second
```

Figure 8.9 Program listing of the project

8.5 Project 3 – Binary counting with 8 LEDs

Description: In this project, eight LEDs are connected to the Raspberry Pi 5 GPIO pins. The LEDs count up in binary every second. The aim of this project is to show how eight LEDs can be connected to the Raspberry Pi 5 GPIO pins. In addition, the project shows how to group the LEDs as an 8-bit port and control them as a single port.

Block diagram: The block diagram of the project is shown in Figure 8.10

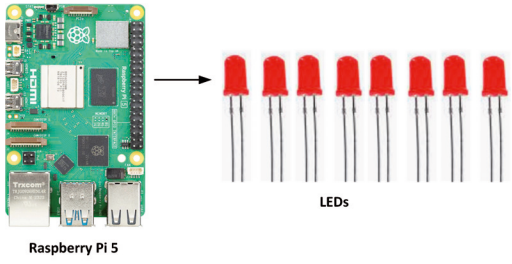

Figure 8.10 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 8.11. The LEDs are connected to 8 GPIO pins through 470 Ω current limiting resistors. The following 8 GPIO pins are grouped as an 8-bit port, where GPIO 2 is configured as the LSB and GPIO 9 is configured as the MSB:

	MSB							**LSB**
GPIO:	9	10	22	27	17	4	3	2
Pin no:	21	19	15	13	11	7	5	3

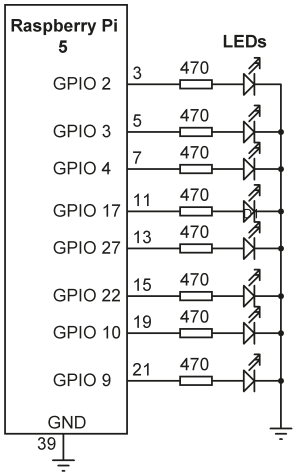

Figure 8.11 Circuit diagram of the project

Construction: The project is constructed on a breadboard as shown in Figure 8.12. Notice that in this project, a **T-Cobbler** (Figure 8.13) connects to the 40-pin GPIO header of the Raspberry Pi through a ribbon cable. On the other side of this ribbon cable, a T-type connector is used which is plugged into a breadboard. This setup simplifies making connections to the Raspberry Pi GPIO header, especially when there are many connections to be made. The GPIO pin names are written on the T-cobbler for ease of access.

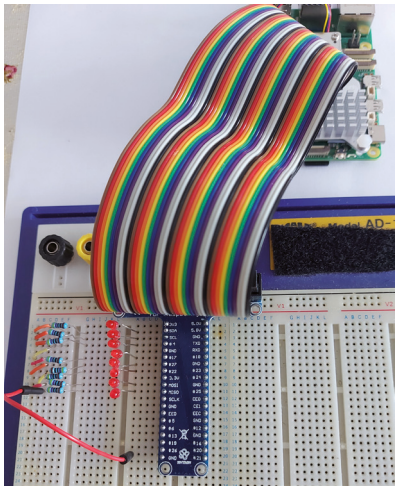

Figure 8.12 Constructing the project on a breadboard

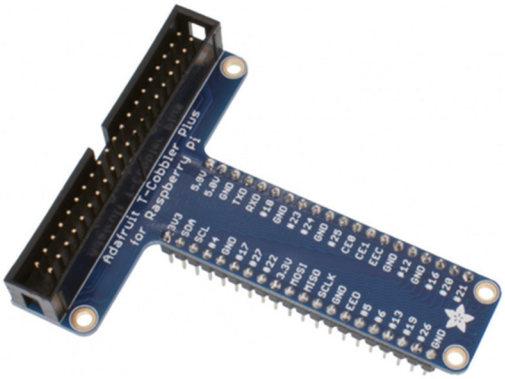

Figure 8.13 The T-Cobbler

Program listing: The program is called **LEDCNT.py** and the listing is shown in Figure 8.14. The program was written using the **nano** text editor. Inside the main program, a loop is formed to execute forever and inside this loop the LEDs count up by one in binary. Variable **cnt** is used as the counter. Function **Port_Output** is used to control the LEDs. This function can take integer numbers from 0 to 255, and it converts the input number (x) into binary using the built-in function **bin**. Then the leading '0b' characters are removed from the output string **b** (**bin** function inserts characters '0b' to the beginning of the converted string). Then, the converted string **b** is made up of 8 characters by inserting leading zeroes. The string is then sent to the PORT bit by bit, starting from the least significant bit (GPIO 2) position. The result is that the 8 LEDs count up in binary.

```
#-------------------------------------------------------------
#
#                      BINARY UP COUNTING LEDs
#                      =======================
#
# In this project 8 LEDs are connected to the following
# GPIO pins:
#
# 9 10 22 27 17 4 3 2
#
# The program groups these LEDs as an 8-bit port and then
# the LEDs count up in binary with one second delay between
# each output.
#
# Program: LEDCNT.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#-------------------------------------------------------------
from gpiozero import LED
from time import sleep         # import time library

#
# LED connections
#
PORT = [0] * 8
PORT[0] = LED(9)
PORT[1] = LED(10)
PORT[2] = LED(22)
PORT[3] = LED(27)
PORT[4] = LED(17)
PORT[5] = LED(4)
PORT[6] = LED(3)
PORT[7] = LED(2)

#
# This function sends 8-bit data (0 to 255) to the PORT
#
def Port_Output(x):
   b = bin(x)                          # convert into binary
   b = b.replace("0b", "")             # remove leading "0b"
   diff = 8 - len(b)                   # find the length
   for i in range (0, diff):
      b = "0" + b                      # insert leading os

   for i in range (0, 8):
      if b[i] == "1":
```

```
         PORT[i].on()                  # bit ON
      else:
         PORT[i].off()                 # bit OFF
   return
#
# Main program loop. Count up in binary every second
#
cnt = 0

while True:
  Port_Output(cnt)                     # send cnt to port
  sleep(1)                             # wait 1 second
  cnt = cnt + 1                        # increment cnt
  if cnt > 255:
     cnt = 0
```

Figure 8.14 Program listing

Recommended modifications: Modify the program such that the LEDs count down every two seconds.

Modified Program

The program shown in Figure 8.14 can be modified and made more friendly by storing the LED port numbers in a list. The modified program **LEDCNT2.py** is shown in Figure 8.15. In this program, the LED port numbers are stored in list PORT. Then function **Port_Output** is used as before to send the port data to the LEDs.

```
#-------------------------------------------------------------
#
#                      BINARY UP COUNTING LEDs
#                      =======================
#
# In this project 8 LEDs are connected to the following
# GPIO pins:
#
# 9 10 22 27 17 4 3 2
#
# The program groups these LEDs as an 8-bit port and then
# the LEDs count up in binary with one second delay between
# each output.
#
# Program: LEDCNT2.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#-------------------------------------------------------------
from gpiozero import LED
```

```
from time import sleep                  # import time library

PORT = [9,10,22,27,17,4,3,2]            # LED ports

#
# This function initializes the port list PORT[]
#
def Configure():
  for i in range(8):
     PORT[i] = LED(PORT[i])

#
# This function sends 8-bit data (0 to 255) to the PORT
#
def Port_Output(x):
   b = bin(x)                           # convert into binary
   b = b.replace("0b", "")              # remove leading "0b"
   diff = 8 - len(b)                    # find the length
   for i in range (0, diff):
      b = "0" + b                       # insert leading os

   for i in range (0, 8):
      if b[i] == "1":
         PORT[i].on()
      else:
         PORT[i].off()
   return
#
# Main program loop. Count up in binary every second
#
cnt = 0
Configure()

while True:
  Port_Output(cnt)                      # send cnt to port
  sleep(1)                              # wait 1 second
  cnt = cnt + 1                         # increment cnt
  if cnt > 255:
     cnt = 0
```

Figure 8.15 Modified program

8.6 Project 4 – Christmas lights (random flashing 8 LEDs)

Description: In this project, eight LEDs are connected to the Raspberry Pi 5 GPIO pins as in Project 3. The LEDs flash randomly every 0.5 seconds, just like fancy Christmas lights. The aim of this project is to show how to generate random numbers between 1 and 255.

The block diagram and circuit diagram of the projects are the same as in Figure 8.10 and Figure 8.11 respectively.

Program listing: The program is called **XMAS.py** and the listing is shown in Figure 8.16. The program was written using the **nano** text editor. At the beginning of the program, the **random** module and other required modules are imported to the program. Then, a loop is formed to execute forever and inside this loop a random number is generated between 1 and 255, and this number is used as an argument to function **Port_Output.** The binary pattern corresponding to the generated number is sent to the port, which turns the LEDs ON or OFF randomly.

```
#------------------------------------------------------------
#
#                      CHRISTMAS LIGHTS
#                      ================
#
# In this project 8 LEDs are connected to the Raspberry Pi 3
# and these LEDs flash randomly at 0.5 second intervals. The
# connections of the LEDs are to the following GPIO pins:
#
# 9 10 22 27 17 4 3 2
#
# The program groups these LEDs as an 8-bit port and then
# generates random numbers between 1 and 255 and turns the
# LEDs ON and OFF depending on the generated number.
#
# Program: XMAS.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#------------------------------------------------------------
from gpiozero import LED
from time import sleep         # import time library
import random                  # import random library

PORT = [9,10,22,27,17,4,3,2]   # LED ports

#
# This function initializes the port list PORT[]
#
def Configure():
  for i in range(8):
     PORT[i] = LED(PORT[i])

#
# This function sends 8-bit data (0 to 255) to the PORT
#
```

```
def Port_Output(x):
   b = bin(x)                          # convert into binary
   b = b.replace("0b", "")             # remove leading "0b"
   diff = 8 - len(b)                   # find the length
   for i in range (0, diff):
      b = "0" + b                      # insert leading os

   for i in range (0, 8):
      if b[i] == "1":
         PORT[i].on()
      else:
         PORT[i].off()
   return
#
# Configure PORTs
#
Configure()

#
# Main program loop. Count up in binary every second
#
while True:
  numbr = random.randint(1, 255)       # generate a random number
  Port_Output(numbr)                   # send cnt to port
  sleep(0.5)                           # wait 0.5 second
```

Figure 8.16 Program listing

Recommended modifications: Modify the program such that 10 LEDs can be connected to the Raspberry Pi 5 and flashed randomly.

8.7 Project 5 – Chasing LEDs

Description: In this project, eight LEDs are connected to the Raspberry Pi 5 GPIO pins as in the previous project. As shown in Figure 8.17, the LEDs rotate (chase each other) from the LSB to MSB with a one-second delay between each output.

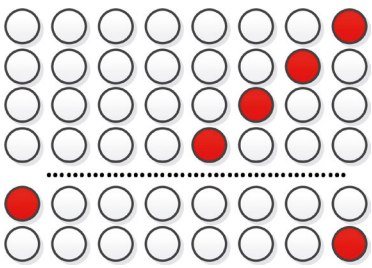

Figure 8.17 Chasing LEDs

The block diagram and circuit diagram of the projects are the same as in Figure 8.10 and Figure 8.11 respectively.

Program listing: The program is called **rotate.py** and the listing is shown in Figure 8.18. The program was written using the **nano** text editor. Inside the main program, a loop is formed to execute forever and inside this loop the variable **rot** is used as an argument to the **Port_Output** function. This variable is shifted left at each iteration, and thus the LED ON sequence is from left to right (from LSB to MSB). A one-second delay is inserted between each output.

```
#--------------------------------------------------------------------
#               ROTATING LEDs
#               =============
#
# In this project 8 LEDs are connected to the Raspberry Pi 5.
# The LEDs rotate from LSB to MSB every second
#
# Program: rotate.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#--------------------------------------------------------------------
from gpiozero import LED
from time import sleep                  # import time library

PORT = [9,10,22,27,17,4,3,2]            # LED ports

#
# This function initializes the port list PORT[]
#
def Configure():
  for i in range(8):
     PORT[i] = LED(PORT[i])

#
# This function sends 8-bit data (0 to 255) to the PORT
#
def Port_Output(x):
   b = bin(x)                           # convert into binary
   b = b.replace("0b", "")              # remove leading "0b"
   diff = 8 - len(b)                    # find the length
   for i in range (0, diff):
      b = "0" + b                       # insert leading os

   for i in range (0, 8):
      if b[i] == "1":
         PORT[i].on()
```

```
      else:
         PORT[i].off()
   return
#
# Configure PORT s
#
Configure()

#
# Main program loop. Rotate the LEDs
#
rot = 1
while True:
  Port_Output(rot)
  sleep(1)                              # wait 1 second
  rot = rot << 1                        # shift left
  if rot > 128:                         # at the end
      rot = 1                           # back to beginning
```

Figure 8.18 Program listing

8.8 Project 6 – Rotating LEDs with push-button switch

Description: In this project, eight LEDs are connected to the Raspberry Pi 5 GPIO pins as in the previous project. In addition, a push-button switch is connected to one of the GPIO ports. The LEDs rotate in one direction when the button is not pressed, and in the opposite direction when the button is pressed. Only one LED is ON at any time. A one-second delay is inserted between each output. The aim of this project is to show how a push-button switch can be connected to a GPIO pin.

Block diagram: The block diagram of the project is shown in Figure 8.19.

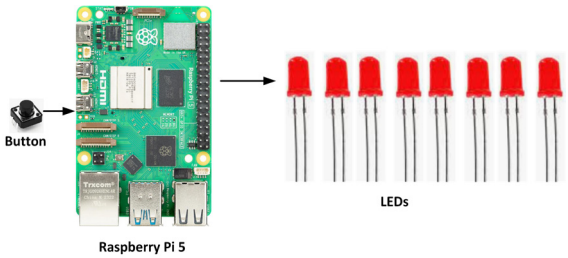

Figure 8.19 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 8.20. The LEDs are connected to 8 GPIO pins through 470 Ω current limiting resistors, as in the previous project. The push-button switch is connected to GPIO 11 (pin 23) of the Raspberry Pi 5. The push-button switch is connected through a 10 kΩ and a 1 kΩ resistor. When the switch is not pressed, the input is at logic 1. When the switch is pressed, the input changes to

logic 0. Notice that the 1 kΩ resistor is used here for safety if the input channel is configured as an output accidentally. If this is the case, without a resistor the output would be short-circuited and this could damage the Raspberry Pi hardware.

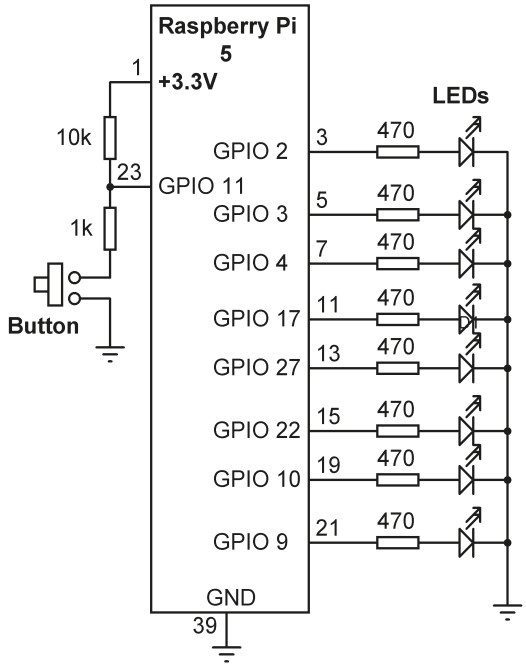

Figure 8.20 Circuit diagram of the project

Construction: The project is constructed on a breadboard as shown in Figure 8.21.

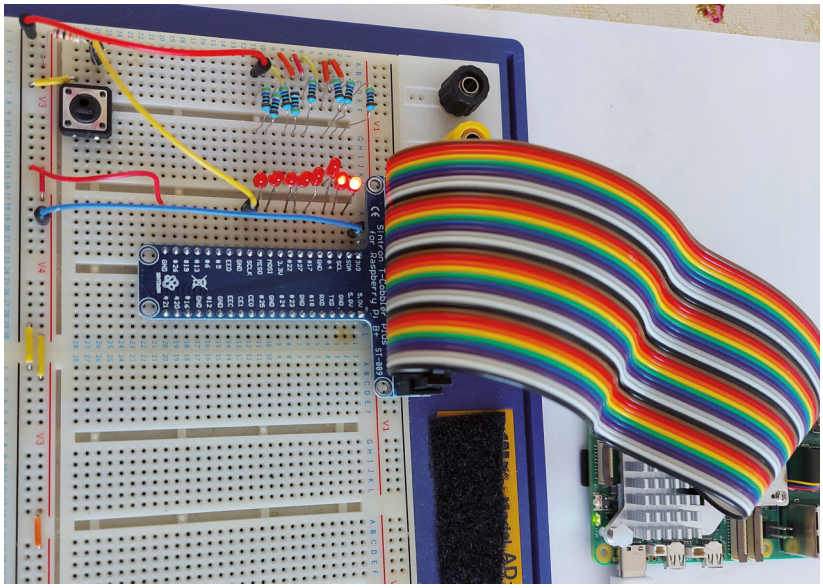

Figure 8.21 Project constructed on a breadboard

Program listing: The program is called **ButtonLED.py** and the listing is shown in Figure 8.22. The program was written using the **nano** text editor. Module **gpiozero** is imported with both **LED** and **Button**. **button** is assigned port GPIO 11. A loop is formed to execute forever, and inside this loop the variable **rot** is used as an argument to the **Port_Output** function. If the button is not pressed, then **rot** is shifted right and the LED ON sequence is from left to right (from MSB to LSB). If, on the other hand, the button is pressed, then the LED On sequence is from right to left (from LSB to MSB). A one-second delay is inserted between each output.

```
#-------------------------------------------------------------------
#
#               ROTATING LEDs WITH PUSH-BUTTON
#               ==============================
#
# In this project 8 LEDs are connected to the Raspberry Pi 5.
# In addition a push-button switch is connected to GPIO 11.
# Normally the output of the button is at logic 1 and goes to 0
# when the button is pressed. The LEDs rotate in one direction
# and when the button is pressed the direction of rotation
# is reversed. One second delay is inserted between each output
#
# Program: ButtonLED.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#-------------------------------------------------------------------
from gpiozero import LED, Button
```

```
from time import sleep                  # import time library

button = Button(11)                     # Button at GPIO 11
PORT = [9,10,22,27,17,4,3,2]            # LED ports

#
# This function initializes the port list PORT[]
#
def Configure():
  for i in range(8):
     PORT[i] = LED(PORT[i])

#
# This function sends 8-bit data (0 to 255) to the PORT
#
def Port_Output(x):
   b = bin(x)                           # convert into binary
   b = b.replace("0b", "")              # remove leading "0b"
   diff = 8 - len(b)                    # find the length
   for i in range (0, diff):
      b = "0" + b                       # insert leading os

   for i in range (0, 8):
      if b[i] == "1":
         PORT[i].on()
      else:
         PORT[i].off()
   return
#
# Configure PORT s
#
Configure()

#
# Main program loop. Rotate the LEDs
#
rot = 1
while True:
  Port_Output(rot)
  sleep(1)
  if button.is_pressed:                 # wait 1 second
     rot = rot << 1                     # shift left
     if rot > 128:                      # at the end
       rot = 1                          # back to beginning
  else:
     rot = rot >> 1
```

```
    if rot == 0:
      rot = 128
```

Figure 8.22 Program listing

Note that the following options are available for the Button function (see https://gpiozero.readthedocs.io/en/stable/api_input.html):

wait_for_press
wait_for_release
held_time
hold_repeat
hold_time
is_held
is_pressed
value
when_held
when_pressed
when_released

8.9 Project 7 – Morse Code exerciser with LED or buzzer

Description: In this project, an LED or a buzzer is connected to GPIO 17 (pin11) of the Raspberry Pi 5. The user enters a text from the keyboard. The buzzer is then turned ON and OFF to sound the letters of the text in Morse code.

Circuit diagram: The circuit diagram of the project is shown in Figure 8.23 where an active buzzer is connected to GPIO 11 of the Raspberry Pi 5.

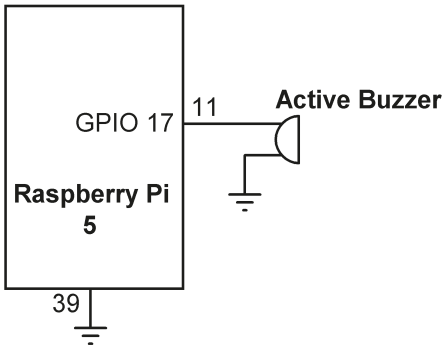

Figure 8.23 Circuit diagram of the project

Morse Code: In Morse code, each letter is made up of dots and dashes. Figure 8.24 shows the Morse code of all the letters in the English alphabet (this table can be extended by adding the Morse code of numbers and punctuation marks). The following rules apply to the timing of dots and dashes:

- The duration of a dot is taken as the unit time, and this determines the speed of the transmission. Normally, the speed of transmission is quoted in words per minute (wpm). The standard required minimum in Morse code communication is 12 wpm.
- The duration of a dash is 3 unit times
- The time between each dot and dash is one unit time
- The time between the letters is 3 unit times
- The time between the words is 7 unit times

The unit time in milliseconds is calculated using the following formula:

Time (ms) = 1200/wpm

In this project, the Morse code is simulated at 10 wpm. Thus, the unit time is taken to be 1200/10 = 120 ms.

Letter	Morse code
A:	.-
B :	-...
C :	-.-.
D :	-..
E :	.
F :	..-.
G :	--.
H :	....
I :	..
J :	.---
K :	-.-
L :	.-..
M :	--
N :	-.
O :	---
P :	.--.
Q :	--.-
R :	.-.
S :	...
T :	-
U :	..-
V :	...-
W :	.--
X :	-..-
Y :	-.--
Z :	--..

Figure 8.24 Morse code of English letters

Program listing: The program is called **morse.py** and the listing is shown in Figure 8.25. The Morse code alphabet is stored in the list **Morse_Code**. The function **DO_DOT** implements a single dot with the duration of one unit time. Function **DO_DASH** implements a single dash with a duration of three unit times. Function **DO_SPACE** implements a space character with duration of seven unit times. The rest of the program is executed in a loop, where a text is read from the keyboard and the buzzer sounds in such a way as to represent the Morse code of this text. The program terminates if the user enters the text **QUIT**.

You should run the program from the command mode as follows:

pi@raspberrypi:~ $ **python morse.py**

```
#-------------------------------------------------------------------
#
#                  MORSE CODE EXERCISER
#                  ====================
#
# This project can be used to learn the Morse code. A buzzer is
# connected to GPIO 17 of the Raspberry Pi 5.
#
# The program reads a text from the keyboard and then sounds the
# buzzer to simulate sending or receiving the Morse code of this
# text.
#
# In this project the Morse code speed is assumed to be 10 wpm,
# but can easily be changed by changing the parameter wpm.
#
# File  : morse.py
# Date  : October, 2023
# Author: Dogan Ibrahim
#-------------------------------------------------------------------
from gpiozero import LED
from time import sleep

Buzzer = LED(17)                        # Buzzer pin

words_per_minute = 10                   # define words per min
wpm = 1200/words_per_minute             # unit time in milliseconds
unit_time = wpm / 1000

Morse_Code = {
         'A': '.-',
         'B': '-...',
         'C': '-.-.',
         'D': '-..',
         'E': '.',
```

```
        'F': '..-.',
        'G': '--.',
        'H': '....',
        'I': '..',
        'J': '.---',
        'K': '-.-',
        'L': '.-..',
        'M': '--',
        'N': '-.',
        'O': '---',
        'P': '.--.',
        'Q': '--.-',
        'R': '.-.',
        'S': '...',
        'T': '-',
        'U': '..-',
        'V': '...-',
        'W': '.--',
        'X': '-..-',
        'Y': '-.--',
        'Z': '--..'
        }

#
# This function sends a DOT (unit time)
#
def DO_DOT():
   Buzzer.on()
   sleep(unit_time)
   Buzzer.off()
   sleep(unit_time)
   return

#
# This function sends a DASH ( 3*unit time)
#
def DO_DASH():
   Buzzer.on()
   sleep(3*unit_time)
   Buzzer.off()
   sleep(unit_time)
   return

#
# This function sends inter-word space (7*unit time)
#
```

```
def DO_SPACE():
   sleep(7*unit_time)
   return

#
# Main program code
#
text = ""
while text != "QUIT":
   text = input("Enter text to send: ")
   if text != "QUIT":
      for letter in text:
         if letter == ' ':
            DO_SPACE()
         else:
            for code in Morse_Code[letter.upper()]:
               if code == '-':
                  DO_DASH()
               elif code == '.':
                  DO_DOT()
               sleep(unit_time)
         sleep(3*unit_time)
   sleep(2)
```

Figure 8.25 Program listing of the project

Recommended modification: An LED can be connected to the GPIO pin instead of the buzzer so that the Morse code can be seen in visual form.

8.10 Project 8 – Electronic dice

Description: In this project seven LEDs are arranged in the form of the faces of a dice and a push-button switch is used. When the button is pressed, the LEDs turn ON to display numbers 1 to 6 as if on a real dice. The display is turned OFF after 3 seconds, ready for the next game. The aim of this project is to show how a dice can be constructed with seven LEDs.

Block diagram: The block diagram of the project is shown in Figure 8.26.

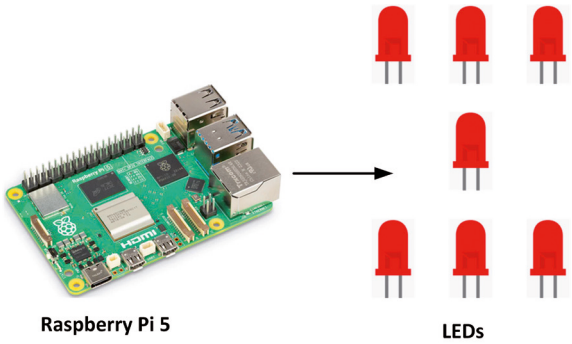

Figure 8.26 Block diagram of the project

Figure 8.27 shows the LEDs that should be turned ON to display the 6 dice numbers.

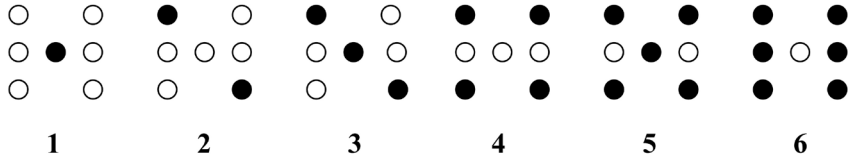

Figure 8.27 LED Dice

Circuit diagram: The circuit diagram of the project is shown in Figure 8.28. Here, 8 GPIO pins are collected together to form a PORT. The following pins are used for the LEDs (there are 7 LEDs, but 8 port pins are used in the form of a byte where the most significant bit position is not used):

Bit	7	6	5	4	3	2	1	0
GPIO:	9	10	22	27	17	4	3	2

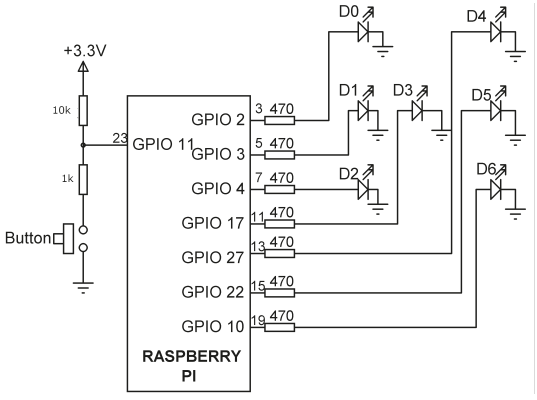

Figure 8.28 Circuit diagram of the project

The push-button switch is connected to port pin GPIO 11.

Table 8.1 gives the relationship between a dice number and the corresponding LEDs to be turned ON to imitate the faces of a real dice. For example, to display number 1 (i.e. only the middle LED is ON), you have to turn LED D3 ON. Similarly, to display number 4, you have to turn ON D0, D2, D4 and D6.

Required number	LEDs to be turned on
1	D3
2	D0, D6
3	D0, D3, D6
4	D0, D2, D4, D6
5	D0, D2, D3, D4, D6
6	D0, D1, D2, D4, D5, D6

Table 8.1 Dice number and LEDs to be turned ON

The relationship between the required number and the data to be sent to the PORT to turn on the correct LEDs is given in Table 8.2. For example, to display dice number 2, you have to send hexadecimal 0x41 to the PORT. Similarly, to display number 5, we have to send hexadecimal 0x5D to the PORT and so on.

Required number	PORT data (Hex)
1	0x08
2	0x41
3	0x49
4	0x55
5	0x5D
6	0x77

Table 8.2 Required number and PORT data

Program listing: The program is called **dice.py** and the listing is shown in Figure 8.29. The bit pattern to be sent to the LEDs corresponding to each dice number is stored in hexadecimal format in a list called DICE_NO (see Table 8.2). GPIO 11 is configured as a button pin, and the push-button switch is connected to this pin to simulate the 'throwing' of a dice. The main program waits until a button is pressed. Then, a random number is generated between 1 and 6 and stored in variable **n**. The bit pattern corresponding to this number is found and sent to function **Port_Output** so that the required LEDs are turned on to represent the dice number. This process is repeated after 3 seconds of delay.

```
#-----------------------------------------------------------------
#
#               ELECTRONIC DICE WITH LEDs
#               =========================
#
# This program is an electronic dice. GPIO 11 of  Raspberry Pi 5
# is configured as a Button. When this button is pressed, a random
# dice number is generated between 1 and 6 and is displayed through
# the LEDs. 7 LEDs are mounted on the breadboard in the form of the
# face of a real dice. The following GPIO pins are used for the LEDs@
#
#  Bit:    7   6   5   4  3  2  1   0
#  GPIO:      10  22  27  17  4  3  2
#
# The following PORT pins are used to construct the dice:
#
# D0     D4
# D1  D3 D5
# D2     D6
#
# Program: dice.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#-----------------------------------------------------------------
from gpiozero import LED, Button
from time import sleep                          # Import time library
import random

button = Button(11)                             # Button at GPIO 11
PORT = [9,10,22,27,17,4,3,2]                    # LED ports
DICE_NO = [0,0x08,0x41,0x49,0x55,0x5D,0x77]

#
# This function initializes the port list PORT[]
#
def Configure():
  for i in range(8):
     PORT[i] = LED(PORT[i])

#
# This function sends 8-bit data (0 to 255) to the PORT
#
def Port_Output(x):
   b = bin(x)                           # convert into binary
   b = b.replace("0b", "")              # remove leading "0b"
   diff = 8 - len(b)                    # find the length
```

```
   for i in range (0, diff):
      b = "0" + b                      # insert leading os

   for i in range (0, 8):
      if b[i] == "1":
         PORT[i].on()
      else:
         PORT[i].off()
   return
#
# Configure PORTs
#
Configure()

#
# Main program loop. Rotate the LEDs
#
while True:
  if button.is_pressed:                        # wait for buttonn
     n = random.randint(1, 6)                  # generate a number
     print(n)
     pattern = DICE_NO[n]
     Port_Output(pattern)
     sleep(3)                                  # wsit for 3 seconds
     Port_Output(0)                            # turn OFF all LEDs
```

Figure 8.29 Program listing of the project

Chapter 9 • Using an I²C LCD

9.1 Overview

The I²C (or I2C) bus is commonly used in microcontroller-based projects. In this chapter, you will be looking at the use of this bus on the Raspberry Pi 5. Some other interesting projects are also given in this chapter. The aim is to make the reader familiar with the I²C bus library functions and to show how they can be used in a real project. Before looking at the details of the projects, it is worthwhile to look at the basic principles of the I²C bus.

9.2 The I²C Bus

I²C bus is one of the most commonly used microcontroller communication protocols for communicating with external devices such as sensors and actuators. The I²C bus is a single master, multiple slave bus, and it can operate at standard mode: 100 Kbit/s, full speed: 400 Kbit/s, fast mode: 1 Mbit/s, and high speed: 3.2 Mbit/s. The bus consists of two open-drain wires, pulled up with resistors:

SDA: data line
SCL: clock line

Figure 9.1 shows the structure of an I²C bus with one master and three slaves.

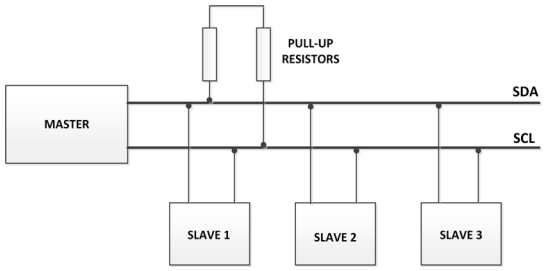

Figure 9.1 I²C bus with one master and three slaves

Because the I²C bus is based on just two wires, there should be a way to address an individual slave device on the same bus. For this reason, the protocol defines that each slave device provides a unique slave address for the given bus. This address is usually 7-bits wide. When the bus is free, both lines are HIGH. All communication on the bus is initiated and completed by the master, which initially sends a START bit, and completes a transaction by sending a STOP bit. This alerts all the slaves that some data is coming on the bus, and all the slaves listen on the bus. After the start bit, 7 bits of unique slave address are sent. Each slave device on the bus has its own address, and this ensures that only the addressed slave communicates on the bus at any time to avoid any collisions. The last sent bit is a read/write bit such that if this bit is 0, it means that the master wishes to write to the bus (e.g. to a register of a slave), if this bit is a 1, it means that the master wishes to read from the bus (e.g. from the register of a slave). The data is sent on the bus with the MSB bit first. An acknowledgement (ACK) bit takes place after every byte and this bit allows the receiver to signal the transmitter that the byte was received successfully and as a result, another byte may be sent. The ACK bit is sent at the 9th clock pulse.

The communication over the I²C bus is simply as follows:

- The master sends on the bus the address of the slave it wants to communicate with
- The LSB is the R/W bit which establishes the direction of data transmission, i.e. from mater to slave (R/W = 0), or from slave to master (R/W = 1)
- Required bytes are sent, each interleaved with an ACK bit, until a stop condition occurs

Depending on the type of slave device used, some transactions may require a separate transaction. For example, the steps to read data from an I²C compatible memory device are:

- Master starts the transaction in write mode (R/W = 0) by sending the slave address on the bus
- The memory location to be retrieved are then sent as two bytes (assuming 64Kbit memory)
- The master sends a STOP condition to end the transaction
- The master starts a new transaction in read mode (R/W = 1) by sending the slave address on the bus
- The master reads the data from the memory. If reading the memory in sequential format, then more than one byte will be read
- The master sets a stop condition on the bus

9.3 I²C pins of Raspberry Pi 5

Raspberry Pi 5 has 2 x I²C pins at its 40-pin GPIO header as follows:

GPIO 2	SDA1	pin 3
GPIO 3	SCL1	pin 5
GPIO 0	SDA0	pin27
GPIO 1	SCL0	pin 28

1.8 Kilo Ohm pull-up resistors are used from the I²C pins to +3.3 V. Notice that because the I²C pins are pulled up to +3.3 V and Raspberry Pi 5 pins are not +5V compatible, it is necessary to use voltage level converter circuits if the I²C LCD operates with +5V.

9.4 Project 1 – Using an I²C LCD – Seconds counter

Description: In this project, an I²C-type LCD is connected to the Raspberry Pi 5. The program counts up in seconds and displays on the LCD. The aim of this project is to show how an I²C-type LCD can be used in Raspberry Pi projects.

The I²C LCD

The I²C LCD has four pins: GND, +V, SDA, and SCL. SDA can be connected to pin GPIO 2 and SCL to pin GPIO 3. +V pin of the display should be connected to the +5 V (pin 2) of the Raspberry Pi 5. Raspberry Pi GPIO pins are not +5 V tolerant, but the I²C LCD operates

with +5 V where its SDA and SCL pins are pulled to +5 V. It is not a good idea to connect the LCD directly to the Raspberry Pi, as it can damage its I/O circuitry. There are several solutions here. One solution is to remove the I²C pull-up resistors on the LCD module. The other option is to use an LCD which operates with +3.3 V. The other solution is to use a bidirectional +3.3 V to +5 V logic level converter chip. In this project, you will use the TXS0102 bidirectional logic level converter chip like the one shown in Figure 9.2.

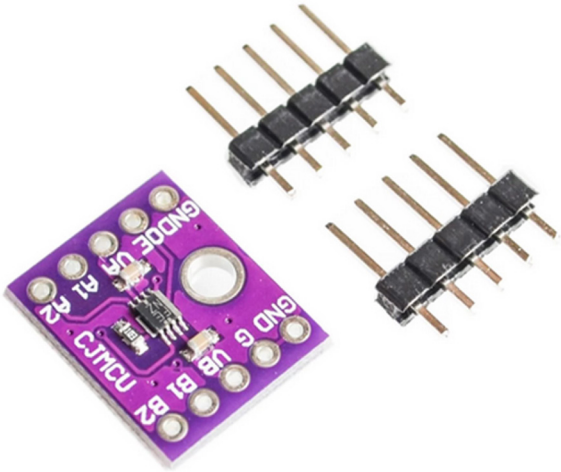

Figure 9.2 Logic-Level converter

Note: Raspberry Pi 5 GPIO pins are claimed to be +5V tolerant as long as the RP1 module is powered ON. But for safety, a logic-level converter is used in this project.

Block diagram: Figure 9.3 shows the block diagram of the project.

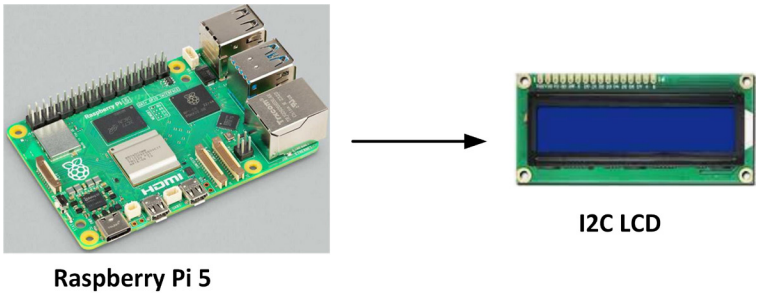

Figure 9.3 Block diagram

Circuit diagram: The circuit diagram is shown in Figure 9.4.

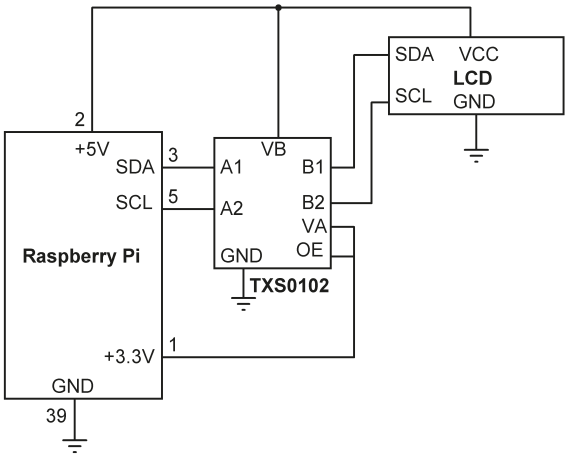

Figure 9.4 Circuit diagram of the project

Figure 9.5 shows the front and back of the I²C based LCD. Notice that the LCD has a small board mounted at its back to control the I²C interface. The LCD contrast is adjusted through the small potentiometer mounted on this board. A jumper is provided on this board to disable the backlight if required.

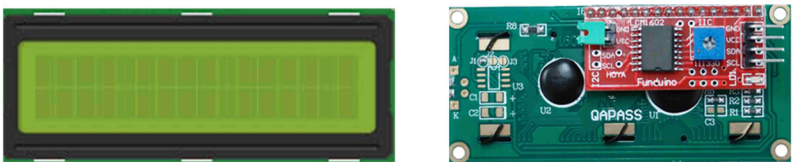

Figure 9.5 I²C based LCD (front and back views)

Program Listing: Before using the I²C pins of the Raspberry Pi, we have to enable the I²C peripheral interface on the device. The steps for this are as follows:

- Start the configuration menu from the command prompt:

  pi@raspberrypi:~ $ **sudo raspi-config**

- Go down the menu to **Interface Options**
- Go down and select **I2C**
- Enable the I²C interface
- Select **Finish** to complete

Now you have to check that the I²C library is available on your Raspberry Pi 5. The steps are as follows:

- Enter the following command. You should see the **I²C** tools (Figure 9.6):

  pi@raspberrypi:~ $ **lsmod | grep i2c**

```
pi@raspberrypi:~ $ lsmod | grep i2c
i2c_designware_platform    65536  0
i2c_designware_core    65536  1 i2c_designware_platform
i2c_dev                65536  0
i2c_brcmstb            65536  0
```

Figure 9.6 Check the I²C tools

- Connect your LCD to the Raspberry Pi 5 device and enter the following command to check whether the LCD is recognized by the Raspberry Pi 5:

  pi@raspberrypi:~ $ **sudo i2cdetect –y 1**

  You should see a table similar to the one shown below. A number in the table means that the LCD has been recognized correctly, and the I²C slave address of the LCD is shown in the table. In this example, the LCD address is 27:

```
     1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --
20:  --  --  --  --  --  --  --  27  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
60:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
70:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
```

You should now install an I²C LCD library so that you can send commands and data to the LCD. There are many Python libraries available for the I²C type LCDs. The one chosen here is on GitHub from Dave Hylands. This library is installed as follows:

- Go to the following web link:

  https://github.com/dhylands/python_lcd/tree/master/lcd

- Copy the following files to your home directory **/home/pi** using WinSCP:

  I2c_lcd.py
  lcd_api.py

- Check to make sure that the file is copied successfully. You should see the file listed with the command:

  pi@raspberrypi: ~ $ **ls**

You are now ready to write the program. Figure 9.7 shows the program listing (**lcd.py**). At the beginning of the program, the LCD driver libraries **lcd_api** and **i2c_lcd** are imported to the program. The heading SECONDS COUNTER is displayed at the top row (row 1) and the program enters a loop. Inside this loop the variable **cnt** is incremented every second and the total value of **cnt** is displayed on the LCD continuously in the following format:

```
SECONDS COUNTER
nn
```

```
#------------------------------------------------------
#             I2C LCD SECONDS COUNTER
#             ======================
#
# In this program an I2C LCD is connected to the Raspberry Pi.
# The program counts up in seconds and displays on the LCD.
#
# At the beginning of the program the text SECONDS COUNTER is
# displayed
#
# Program: lcd.py
# Date   : October 2017
# Author : Dogan Ibrahim
#------------------------------------------------------
import time
from lcd_api import LcdApi
from i2c_lcd import I2cLcd

I2C_ADDR = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

mylcd = I2cLcd(1,I2C_ADDR,I2C_NUM_ROWS,I2C_NUM_COLS)

mylcd.clear()                                  # clear LCD
mylcd.putstr("SECONDS COUNTER")                # display string

cnt = 0                                        # initialize cnt

while True:                                    # infinite loop
  cnt = cnt + 1                                # increment count
  mylcd.move_to(0,1)
  mylcd.putstr(str(cnt))                       # display cnt
  time.sleep(1)                                # wait one second
```

Figure 9.7 Program listing

The I²C LCD library supports many functions. Some of the commonly used functions are (see the LCD library documentation for more details):

clear()	clear LCD and set to home position
show_cursor()	show cursor
hide_cursor()	hide cursor
blink_cursor_on()	blink cursor
blink_cursor_off()	stop blinking cursor
display_on()	display on
display_off()	display off
backlight_on()	backlight on
backlight_off()	backlight off
move_to(x, y)	move cursor to (x, y)
putchar()	display a character
putstr()	display a string

9.5 Project 2 – Using an I²C LCD – Display time

Description: In this project an I²C type LCD is connected to the Raspberry Pi 5 as in the previous project. The program displays the current time on the LCD.

The block diagram and circuit diagram are as in Figure 9.3 and Figure 9.4 respectively.

Program listing: Figure 9.8 shows the program listing (**LCDtime.py**). At the beginning of the program, **time**, **datetime**, and **I2cLCD** modules are imported to the program. The LCD is cleared, and the program enters a loop. Inside this loop, the current time is extracted using the **strftime()** function and the current time is then displayed on the top row of the LCD every second in the following format:

hh:mm:ss

```
#------------------------------------------------------------
#             I2C LCD TIME DISPLAY
#             ======================
#
# This program displays the current time on the LCD.
#
# Program: LCDtime.py
# Date   : October 2017
# Author : Dogan Ibrahim
#------------------------------------------------------------
from time import sleep
from datetime import datetime
from  lcd_api  import LcdApi
from i2c_lcd import I2cLcd

I2C_ADDR = 0x27
```

```
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

mylcd = I2cLcd(1,I2C_ADDR,I2C_NUM_ROWS,I2C_NUM_COLS)
mylcd.clear()                          # clear LCD

while True:                            # infinite loop
  now = datetime.now()
  time = now.strftime("%H:%M:%S")
  mylcd.move_to(0,0)
  mylcd.putstr(str(time))
  sleep(1)                             # wait one second
  mylcd.clear()
```

Figure 9.8 Program listing

9.6 Project 3 – Using an I²C LCD – Display IP address of Raspberry Pi 5

Description: In this project an I²C type LCD is connected to the Raspberry Pi 5 as in the previous projects. The IP address of the Raspberry Pi 5 is displayed on the top row of the LCD.

The block diagram and circuit diagram are as in Figure 9.3 and Figure 9.4 respectively.

Program listing: Figure 9.9 shows the program listing (**LCDip.py**). The IP address is extracted using the **hostname** command with the **–I** option. The IP address is then displayed on the LCD in the following format:

192.168.3.196

```
#-----------------------------------------------------------
#
#             I2C LCD IP DISPLAY
#             ==================
# This program displays the IP address on the LCD.
#
# Program: LCDip.py
# Date   : October 2017
# Author : Dogan Ibrahim
#-----------------------------------------------------------
from time import sleep
from subprocess import check_output
from  lcd_api import LcdApi
from i2c_lcd import I2cLcd
```

```
I2C_ADDR = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

mylcd = I2cLcd(1,I2C_ADDR,I2C_NUM_ROWS,I2C_NUM_COLS)
mylcd.clear()

ip = check_output(["hostname", "-I"],encoding="utf-8").split()[0]
mylcd.putstr(str(ip))

while True:
   pass
```

Figure 9.9 Program listing

9.7 Project 4 – Voltmeter – Output to the screen

Description: This is a voltmeter project. Because the Raspberry Pi 5 does not have any analog-to-digital converters (ADC) on-board, an external ADC chip is used in this project. The voltage to be measured is applied to the ADC and its value is displayed on the screen.

Block diagram: Figure 9.10 shows the block diagram.

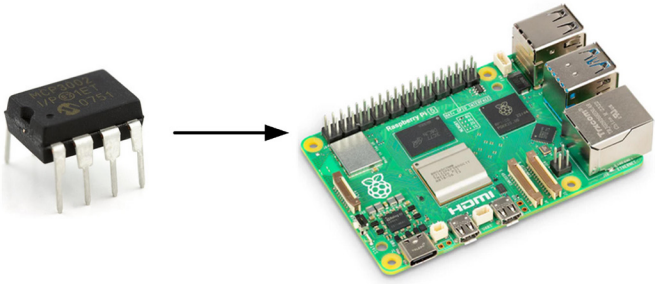

Figure 9.10 Block diagram

Circuit Diagram: The dual MCP3002 ADC chip is used in this project to provide analog input capability to the Raspberry Pi 5. This chip has the following features:

- 10-bit resolution (0 to 1023 quantization levels)
- On-chip sample and hold
- SPI bus compatible
- Wide operating voltage (+2.7 V to +5.5 V)
- 75 KSPS sampling rate
- 5 nA standby current, 50 µA active current

The MCP3002 is a successive-approximation 10-bit ADC with on-chip sample-and-hold amplifier. The device is programmable to operate as either a differential input pair or as dual single-ended inputs. The device is offered in an 8-pin package. Figure 9.11 shows the pin configuration of the MCP3002.

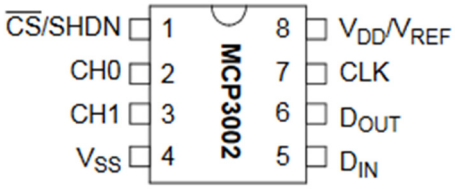

Figure 9.11 Pin configuration of the MCP3002

The pin definitions are as follows:

Vdd/Vref:	Power supply and reference voltage input
CH0:	Channel 0 analog input
CH1:	Channel 1 analog input
CLK:	SPI clock input
DIN:	SPI serial data in
DOUT:	SPI serial data out
CS/SHDN:	Chip select/shutdown input

In this project, the supply voltage and the reference voltage are set to +3.3 V. Thus, the digital output code is given by:

Digital output code = 1024 x Vin / 3.3

or, Digital output code = 310.30 x Vin

Each quantization level corresponds to 3300/1024 = 3.22 mV. Thus, for example, input data '00 0000001' corresponds to 3.22 mV, '00 0000010' corresponds to 6.44 mV and so on.

The MCP3002 ADC has two configuration bits: SGL/DIFF and ODD/SIGN. These bits follow the sign bit and are used to select the input channel configuration. The SGL/DIFF is used to select single ended or pseudo-differential mode. The ODD/SIGN bit selects which channel is used in single-ended mode and is used to determine polarity in pseudo-differential mode. In this project, we are using channel 0 (CH0) in single-ended mode. According to the MCP3002 data sheet, SGL/DIFF and ODD/SIGN must be set to 1 and 0 respectively.

Figure 9.12 shows the circuit diagram of the project, where the voltage to be measured is applied directly to the CH0 input of the ADC. MCP3002 operates with the SPI interface. Raspberry Pi 5 GPIO SPI pins are:

SPI0:
MISO - pin 21
MOSI - pin 19
CE0 - pin 24
SCLK - pin 23

SPI1:
MISO - pin 35
MOSI - pin 38
CE1 - pin 11
SCLK - pin 40

In this project, SPI0 GPIO pins are used.

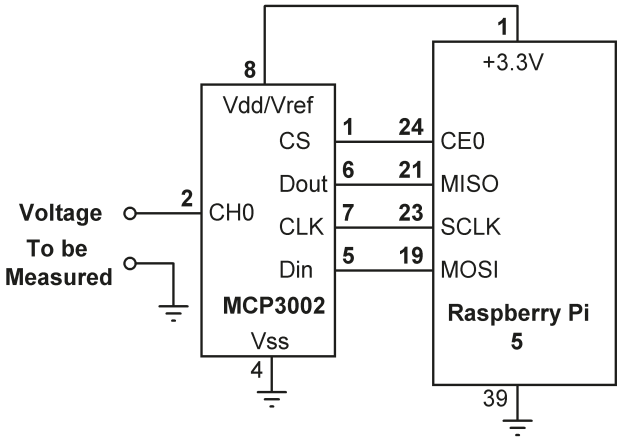

Figure 9.12 Circuit diagram of the project

Program listing: The SPI interface must be enabled on the Raspberry Pi 5 before using the SPI functions. The steps are:

- Start the configuration menu from the command prompt:

  pi@raspberrypi:~ $ **sudo raspi-config**

- Go down the menu to **Interface Options**
- Go down and select **SPI**
- Enable the SPI interface
- Select **Finish** to complete

Figure 9.13 shows the program listing (**voltmeter.py**). The function **get_adc_data** is used to read the analog data, where the channel number (channel_no) is specified in the function argument as 0 or 1. Notice that we have to send the start bit, followed by the SGL/DIFF and ODD/SIGN bits and the MSBF bit to the chip.

It is recommended to send leading zeroes on the input line before the start bit. This is often done when using microcontroller-based systems that must send 8 bits at a time.

The following data can be sent to the ADC (SGL/DIFF = 1 and ODD/SIGN = channel_no) as bytes with leading zeroes for more stable clock cycle. The general data format is:

0000 000S DCM0 0000 0000 0000

Where, S = start bit, D = SGL/DIFF bit, C = ODD/SIGN bit, M = MSBF bit

For channel 0: 0000 0001 1000 0000 0000 0000 (0x01, 0x80, 0x00)

For channel 1: 0000 0001 1100 0000 0000 0000 (0x01, 0xC0, 0x00)

Notice that the second byte can be sent by adding 2 to the channel number (to make it 2 or 3) and then shifting 6 bits to the left as shown above to give 0x80 or 0xC0.

The chip returns 24-bit data (3 bytes) and we must extract the correct 10-bit ADC data from these 24 bits. The 24-bit data is in the following format ('X' is don't care bit):

XXXX XXXX XXXX DDDD DDDD DDXX

Assuming that the returned data is stored in 24-bit variable ADC, we have:

ADC[0] = "XXXX XXXX"
ADC[1] = "XXXX DDDD"
ADC[2] = "DDDD DDXX"

Thus, we can extract the 10-bit ADC data with the following operations:

(ADC[2] >> 2) so, low byte = '00DD DDDD'
and
(ADC[1] & 15) << 6) so, high byte = 'DD DD00 0000'

Adding the low byte and the high byte, we get the 10-bit converted ADC data as:

DD DDDD DDDD

The SPI bus on the Raspberry Pi supports the following functions:

Function	Description
open (0,0)	Open SPI bus 0 using CE0
open (0,1)	Open SPI bus 0 using CE1
close()	disconnect the device from the SPI bus
writebytes([array of bytes])	Write an array of bytes to SPI bus device
readbytes(len)	Read **len** bytes from SPI bus device
xfer2([array of bytes])	Send an array of bytes to the device with CEx asserted at all times
xfer([array of bytes])	Send an array of bytes de-asserting and asserting CEx with every byte transmitted

At the beginning of the program in Figure 9.13 an instance of the SPI is created. Function **get_adc_data** reads the temperature from the sensor chip MCP3002 and returns a digital value between 0 and 1023. This value is then converted into millivolts and is displayed on the screen. Figure 9.14 shows an example output from the project where the input CH0 was connected to GND or to +3.3 V.

```
#-----------------------------------------------------------------
#                              VOLTMETER
#                              =========
#
# This is a voltmeter project. The voltage to be measured is applied
# to CH0 input of the MCP3002 ADC. The measured voltage is displayed
# on the screen using a print statement
#
# Program: voltmeter.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#-----------------------------------------------------------------
import spidev
from time import sleep

#
# Create SPI instance and open the SPI bus
#
spi = spidev.SpiDev()
spi.open(0,0)                          # we are using CE0 for CS
spi.max_speed_hz = 4000

#
# This function returns the ADC data read from the MCP3002
#
def get_adc_data(channel_no):
   ADC = spi.xfer2([1, (2 + channel_no) << 6, 0])
   rcv = ((ADC[1] & 15) << 6) + (ADC[2] >> 2)
   return rcv
```

```
#
# Start of main program. Read the analog temperature, convert
# into degrees Centigrade and display on the monitor every second
#
while True:
    adc = get_adc_data(0)
    mV = adc * 3300.0 / 1023.0          # convert to mV
    print("Voltage = %5.2f mV" %mV)     # display voltage in mV
    sleep(1)                            # wait one second
```

Figure 9.13 Program listing

```
pi@raspberrypi:~ $ python voltmeter.py
Voltage = 3287.10 mV
Voltage = 3296.77 mV
Voltage = 3290.32 mV
Voltage = 3290.32 mV
Voltage = 3274.19 mV
Voltage =  0.00 mV
Voltage =  0.00 mV
Voltage =  0.00 mV
Voltage =  0.00 mV
Voltage = 3300.00 mV
Voltage = 3280.65 mV
Voltage = 3277.42 mV
Voltage = 3290.32 mV
```

Figure 9.14 Example output from the program

9.8 Project 5 – Voltmeter – Output to LCD

Description: This project is basically the same as the previous one, but here the measured voltage is displayed on LCD.

Block diagram: Figure 9.15 shows the block diagram.

Figure 9.15 Block diagram

Circuit Diagram: The circuit diagram of the project is shown in Figure 9.16. The LCD and the MCP3002 are connected as in the previous projects.

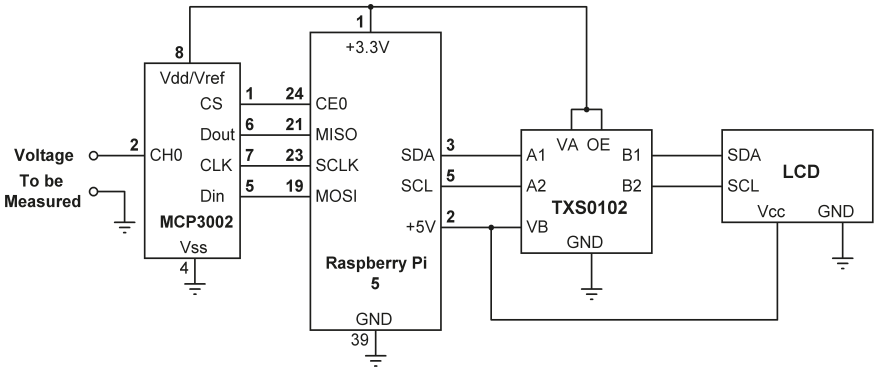

Figure 9.16 Circuit diagram of the project

Program listing: Figure 9.17 shows the program listing (**LCDvolt.py**). This program is basically the same as the one in Figure 9.13, but here the output is sent to LCD instead of being displayed on the screen. The data is displayed in the following format:

nnnn mV

```
#----------------------------------------------------------------
#                      VOLTMETER WITH LCD DISPLAY
#                      ==========================
#
# This is a voltmeter project. The voltage to be measured is applied
# to CH0 input of the MCP3002 ADC. The measured voltage is displayed
# on the LCD
#
# Program: LCDvolt.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#----------------------------------------------------------------
import spidev
from lcd_api import LcdApi
from i2c_lcd import I2cLcd
from time import sleep

#
# Create SPI instance and open the SPI bus
#
spi = spidev.SpiDev()
spi.open(0,0)                          # we are using CE0 for CS
spi.max_speed_hz = 4000

I2C_ADDR = 0x27
```

```
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16


mylcd = I2cLcd(1,I2C_ADDR,I2C_NUM_ROWS,I2C_NUM_COLS)
mylcd.clear()

#
# This function returns the ADC data read from the MCP3002
#
def get_adc_data(channel_no):
   ADC = spi.xfer2([1, (2 + channel_no) << 6, 0])
   rcv = ((ADC[1] & 15) << 6) + (ADC[2] >> 2)
   return rcv

#
# Start of main program. Read the analog temperature, convert
# into degrees Centigrade and display on the monitor every second
#
while True:
   adc = get_adc_data(0)
   mV = adc * 3300.0 / 1023.0                  # convert to mV
   disp = str(mV)[:4] + " mV"
   mylcd.move_to(0,0)
   mylcd.putstr(disp)
   sleep(2)
   mylcd.clear()
```

Figure 9.17 Program listing

9.9 Project 6 – Analog temperature sensor thermometer – output to the screen

Description: In this project, an analog temperature sensor chip is used to measure and then display the ambient temperature every second on the screen. The temperature is read using an external ADC, as in the previous project. The aim of this project is to show how the ambient temperature can be read and displayed on the monitor using an analog temperature sensor chip.

Block Diagram: Figure 9.18 shows the block diagram of the project.

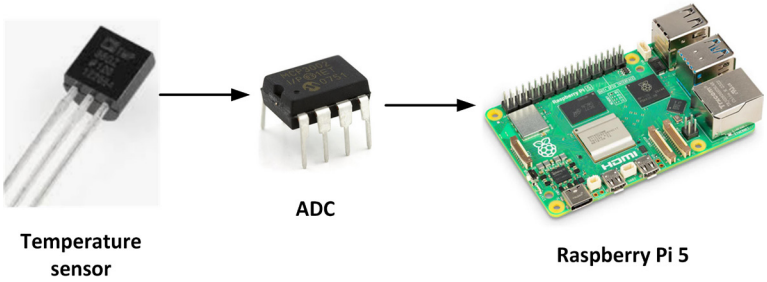

Figure 9.18 Block diagram of the project

Circuit Diagram: The dual MCP3002 ADC chip is used in this project to provide analog input capability to the Raspberry Pi. Figure 9.19 shows the circuit diagram of the project. A TMP36DZ type analog temperature sensor chip is connected to CH0 of the ADC. TMP36DZ is a 3-terminal small sensor chip with pins: Vs, GND, and Vo. Vs is connected to +3.3 V, GND is connected to system ground, and Vo is the analog output voltage. The temperature in degrees centigrade is given by:

Temperature = (Vo − 500) / 10

Where, Vo is the sensor output voltage in millivolts.

CS, Dout, CLK, and Din pins of the ADC are connected to the SPI pins CE0, MISO, SCLK, and MOSI pins of the Raspberry Pi 5 respectively.

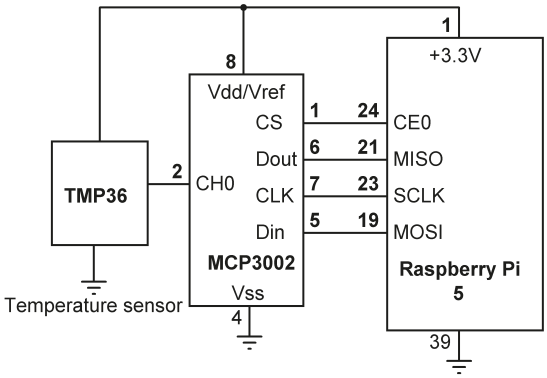

Figure 9.19 Circuit diagram of the project

Program listing: Figure 9.20 shows the Raspberry Pi Python program listing (program: **tmp36.py**). The function **get_adc_data** is used to read the analog data, where the channel number (channel_no) is specified in the function argument as 0 or 1. Function **get_adc_data** reads the temperature from the sensor chip MCP3002 and returns a digital value between 0 and 1023. This value is then converted into millivolts, 500 is subtracted from it, and the result is divided by 10 to find the temperature in degrees centigrade. The temperature is displayed on the monitor every second.

```
#-----------------------------------------------------------------
#               ANALOG TEMPERATURE MEASUREMENT
#               ==============================
#
# This is a thermometer project. Ambient temperature is read using
# an ADC and is then displayed on the screen every second
#
# Program: tmp36.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#-----------------------------------------------------------------
import spidev
from time import sleep

#
# Create SPI instance and open the SPI bus
#
spi = spidev.SpiDev()
spi.open(0,0)                          # we are using CE0 for CS
spi.max_speed_hz = 4000

#
# This function returns the ADC data read from the MCP3002
#
def get_adc_data(channel_no):
   ADC = spi.xfer2([1, (2 + channel_no) << 6, 0])
   rcv = ((ADC[1] & 15) << 6) + (ADC[2] >> 2)
   return rcv

#
# Start of main program. Read the analog temperature, convert
# into degrees Centigrade and display on the monitor every second
#
while True:
   adc = get_adc_data(0)
   mV = adc * 3300.0 / 1023.0                 # convert to mV
   Temperature = (mV - 500) / 10.0
   print("Temperature = %5.2f C" %Temperature)
   sleep(1)                                   # wait one second
```

Figure 9.20 Python program listing

A typical display on the monitor is shown in Figure 9.21.

```
pi@raspberrypi:~ $ python tmp36.py
Temperature = 20.00 C
Temperature = 20.00 C
Temperature = 20.00 C
Temperature = 20.00 C
Temperature = 20.00 C
Temperature = 20.00 C
Temperature = 20.00 C
Temperature = 20.00 C
```

Figure 9.21 Typical display

9.10 Project 7 – Analog temperature sensor thermometer – output on LCD

Description: This project is similar to the previous one, but here the temperature is displayed on LCD.

Block diagram: Figure 9.22 shows the block diagram of the project.

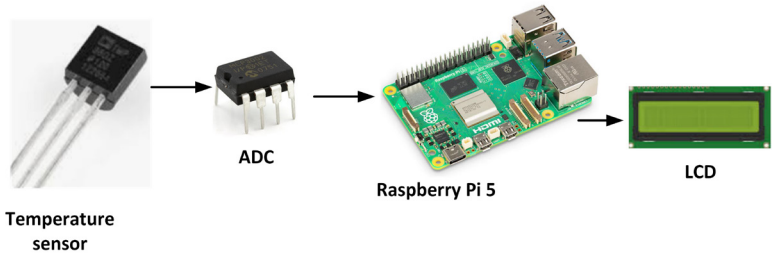

Figure 9.22 Block diagram

Circuit diagram: The circuit diagram of the project is shown in Figure 9.23. The ADC and the sensor chip are connected as in the previous project.

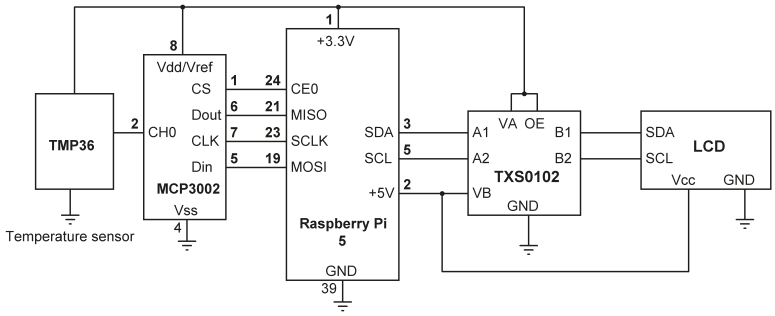

Figure 9.23 Circuit diagram

Program listing: Figure 9.24 shows the program listing (**LCDtmp36.py**). The program is very similar to the previous one, but here the temperature is displayed on LCD.

```
#-----------------------------------------------------------------
#           ANALOG TEMPERATURE MEASUREMENT - OUTPUT ON LCD
#           ==============================================
#
# This is a thermometer project. Ambient temperature is read using
# an ADC and is then displayed on LCD
#
# Program: LCDtmp36.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#-----------------------------------------------------------------
import spidev
from time import sleep
from lcd_api import LcdApi
from i2c_lcd import I2cLcd

I2C_ADDR = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

mylcd = I2cLcd(1,I2C_ADDR, I2C_NUM_ROWS,I2C_NUM_COLS)
mylcd.clear()

#
# Create SPI instance and open the SPI bus
#
spi = spidev.SpiDev()
spi.open(0,0)                          # we are using CE0 for CS
spi.max_speed_hz = 4000

#
# This function returns the ADC data read from the MCP3002
#
def get_adc_data(channel_no):
   ADC = spi.xfer2([1, (2 + channel_no) << 6, 0])
   rcv = ((ADC[1] & 15) << 6) + (ADC[2] >> 2)
   return rcv

#
# Start of main program. Read the analog temperature, convert
# into degrees Centigrade and display on the monitor every second
#
while True:
   adc = get_adc_data(0)
   mV = adc * 3300.0 / 1023.0          # convert to mV
   Temperature = (mV - 500) / 10.0
```

```
T = str(Temperature)[:5] + " C"
mylcd.move_to(0,0)
mylcd.putstr(T)
sleep(5)                              # wait one second
mylcd.clear()
```

Figure 9.24 Program listing

9.11 Project 8 – Reaction timer – output to screen

Description: This is a reaction timer project. The user presses a button as soon as he/she sees a LED lighting. The time delay between seeing the light and pressing the button is measured and displayed on the screen. The LED then turns OFF and the process is repeated after a random delay of 1 to 10 seconds. The aim of this project is to show how the time can be read and how a simple reaction timer project can be designed.

Block Diagram: Figure 9.25 shows the block diagram of the project.

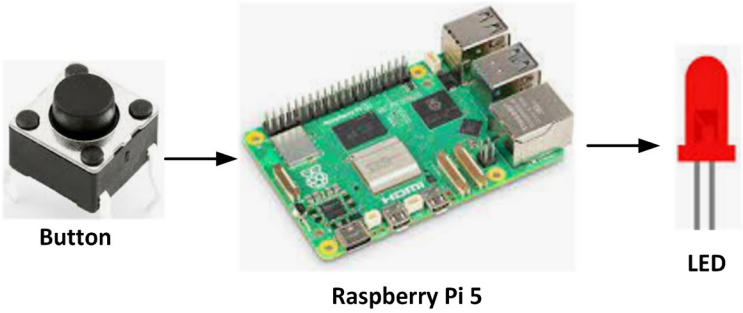

Figure 9.25 Block diagram of the project

Circuit Diagram: The circuit diagram of the project is basic, and it consists of an LED and a push-button switch. The LED and the button are connected to GPIO 17 and GPIO 3 respectively. The button is connected using two resistors as shown in Figure 9.26.

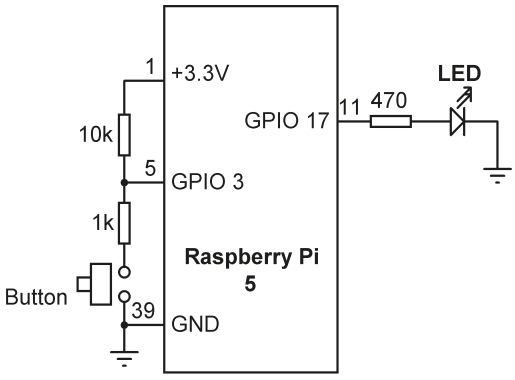

Figure 9.26 Circuit diagram of the project

Program listing: The program is called **reaction.py** and its listing is shown in Figure 9.27. At the beginning of the program, the random library and other used libraries are imported to the program. The program runs in a loop where the system time is recorded as soon as the LED is turned ON. The program waits for the user to press the button, and the system time is read again at this moment. The difference between this time and the first time is displayed as the reaction time of the user. This process repeats after a random delay of 1 to 10 seconds. Notice that the floating-point function **time.time()** returns the time in seconds since the epoch.

```
#-------------------------------------------------------------
#
#                  REACTION TIMER
#                  ==============
#
# This is a reaction timer program. The user presses a button
# as soon as he/she see a light. The time between seeing the
# light and pressing the button is measured and is displayed
# in milliseconds as the reaction time of the user. The light
# comes ON after a random number of seconds between 1 and 10
# seconds.
#
# Program: reaction.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#-------------------------------------------------------------
from time import sleep
import random
from gpiozero import LED, Button
import time

button = Button(3)                      # At GPIO 3
led = LED(17)                           # At GPIO 17

# Start of main program
#
while True:
   T = random.randint(1, 10)
   sleep(T)
   led.on()                             # LED ON
   start_time = time.time()             # start time
   button.wait_for_press()              # wait for button
   end_time = time.time()
   diff_time = 1000.0*(end_time - start_time)
   diff_int = int(diff_time)
   print("Reaction time=%d ms" %diff_int)
   led.off()                            # LED OFF
   sleep(3)
```

Figure 9.27 Program listing

An example output is shown in Figure 9.28.

```
pi@raspberrypi:~ $ python reaction.py
Reaction time=4077 ms
Reaction time=1577 ms
Reaction time=847 ms
Reaction time=327 ms
Reaction time=845 ms
```

Figure 9.28 Example output

9.12 Project 9 – Reaction timer – output to LCD

Description: This project is very similar to the previous one, but here the output is sent to LCD instead of the screen. As before, the user presses a button as soon as he/she sees an LED lighting. The time delay between seeing the light and pressing the button is measured and displayed on the LCD. The LED then turns OFF and the process is repeated after a random delay of 1 to 10 seconds.

Block Diagram: Figure 9.29 shows the block diagram of the project.

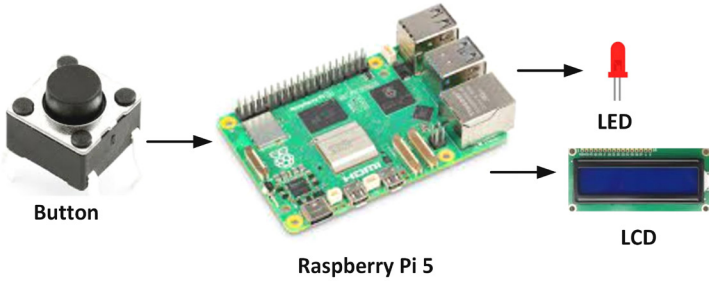

Figure 9.29 Block diagram of the project

Circuit Diagram: The circuit diagram of the project, shown in Figure 9.30, is simple, and it consists of an LED, a push-button switch, and an LCD. The LED and the button are connected to GPIO 17 and GPIO 4 respectively.

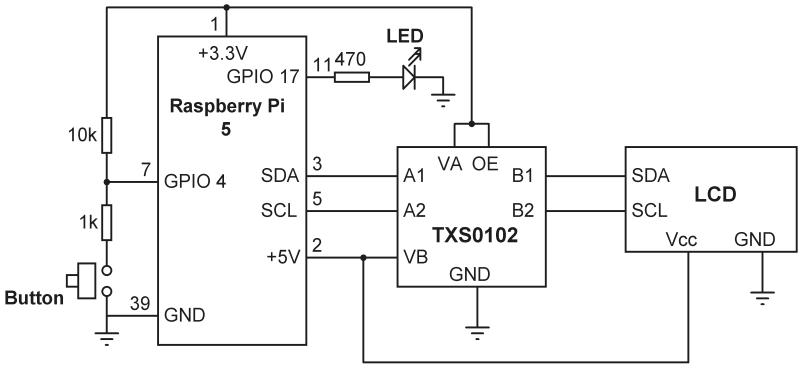

Figure 9.30 Circuit diagram of the project

Program listing: The program is called **LCDreaction.py** and its listing is shown in Figure 9.31. The program is basically the same as the one in Figure 9.27, but here the output is sent to LCD.

```
#-------------------------------------------------------------
#
#                 REACTION TIMER - OUTPUT TO LCD
#                 ==============================
#
# This is a reaction timer program. The user presses a button
# as soon as he/she see a light. The time between seeing the
# light and pressing the button is measured and is displayed
# on LCD in milliseconds as the reaction time of the user. The
# light comes ON after a random number of seconds between 1 and
# 10 seconds.
#
# Program: LCDreaction.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#-------------------------------------------------------------
from time import sleep
import random
from gpiozero import LED, Button
import time
from  lcd_api import  LcdApi
from i2c_lcd import  I2cLcd

I2C_ADDR = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

mylcd = I2cLcd(1, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)
mylcd.clear()

button = Button(4)                      # At GPIO 4
led = LED(17)                           # At GPIO 17

# Start of main program
#
while True:
   T = random.randint(1, 10)
   sleep(T)
   led.on()                             # LED ON
   start_time = time.time()             # start time
   button.wait_for_press()              # wait for button
   end_time = time.time()
```

```
diff_time = 1000.0*(end_time - start_time)
diff_int = int(diff_time)
mylcd.move_to(0, 0)
mylcd.putstr(diff_int)
led.off()                          # LED OFF
sleep(3)
mylcd.clear()
```

Figure 9.31 Program listing

9.13 Project 10 – Automatic dusk lights

Description: In this project, a light dependent resistor (LDR) is used to sense the darkness and a relay is activated when the ambient light intensity falls below the required level. It is possible to connect e.g. lights to the relay so that they turn ON automatically when, for example, it is dusk. The aim of this project is to show how to use an LDR in a Raspberry Pi project, and also how to connect and activate a relay.

Block Diagram: Figure 9.32 shows the block diagram of the project.

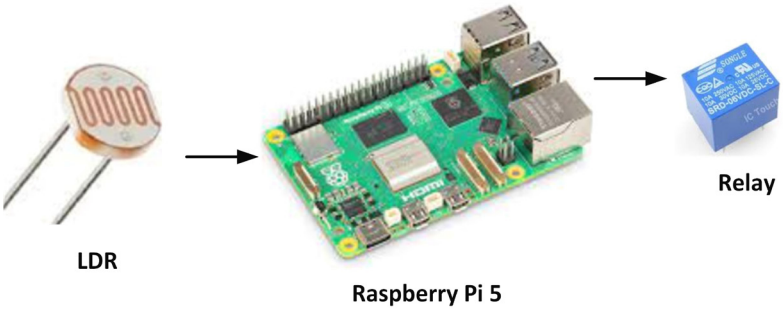

Figure 9.32 Block diagram of the project

Circuit Diagram: As shown in Figure 9.33, the circuit diagram of the project is simple, and it consists of an LDR, a 10 kΩ potentiometer, and a relay. The LDR is connected to GPIO 4, and the relay to GPIO 17.

The resistance of an LDR increases as the light level falls. The response of a typical LDR is shown in Figure 9.34. The LDR is connected as a resistive potential divider circuit. The voltage across the LDR increases as the light level falls. At dark, logic 0 will be sent to the Raspberry Pi, which in turn will activate the relay. When it is light, logic 1 will be sent to the Raspberry Pi, which will deactivate the relay. The potentiometer can be adjusted so that the relay is activated at the required light level. This process will require some trial and error.

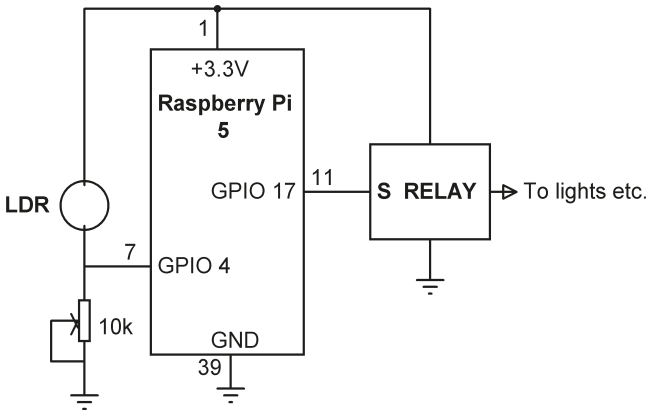

Figure 9.33 Circuit diagram of the project

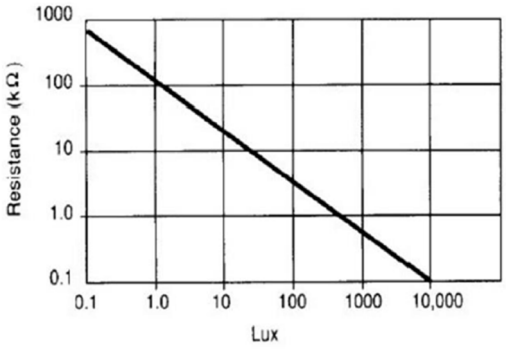

Figure 9.34 Response of a typical LDR

Program listing: Figure 9.35 shows the program listing (program: **dusklight.py**). The LDR is input and the relay is output. The program detects the voltage at its GPIO 4 pin and if it at logic 0 (i.e. dark) then it deactivates the relay, otherwise the relay activated. The potentiometer can be used to adjust the required light trigger level.

```
#-------------------------------------------------------------
#
#                   DUSK LIGHT
#                   ==========
#
# In this project a light dependent resistor (LDR) is used to
# detect the ambient light level. When the light level falls
# below the required value, a relay is activated which turns
# ON the lights.
#
# The potentiometer can be used to adjust the triggering
# light level of the project.
#
```

```
# Program: dusklight.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#-------------------------------------------------------------
from gpiozero import LED, Button

LDR = Button(4)                         # LDR at GPIO 4
RELAY = LED(17)                         # RELAY at GPIO 17

RELAY.off()                     # RELAY OFF)

while True:
   if LDR.is_pressed:
      RELAY.on()                # At logic 0 (dark)
   else:
      RELAY.off()               # At logic 1 (light)
```

Figure 9.35 Program listing

9.14 Project 11 – Ultrasonic distance measurement

Description: This project uses an ultrasonic transmitter/receiver pair to measure the distance in front of the sensor. The distance is displayed on the screen. The aim of the project is to show how ultrasonic sensors can be attached to a Raspberry Pi 5 and how distance can be measured using these sensors.

Block diagram: Figure 9.36 shows the block diagram of the project.

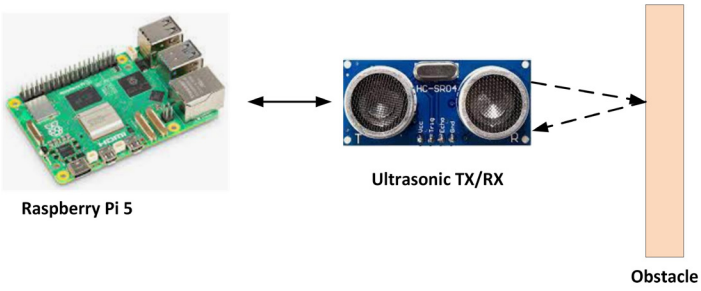

Figure 9.36 Block diagram of the project

Circuit Diagram: An ultrasonic sensor is used to sense the distance in front of the sensor. The outputs of the ultrasonic sensors are +5 V and therefore are incompatible with the inputs of Raspberry Pi 5. A resistive potential divider circuit is used to lower the voltage to +3.3 V. The voltage at the output of the potential divider resistor is:

$$Vo = 5 \times 2k / (2k + 1k) = 3.3 \text{ V}$$

In this project, an HC-SR04-type ultrasonic transmitter/receiver module is used (see Figure 9.37). These modules have the following specifications:

- Operating voltage (current): 5 V (2 mA) operation
- Detection distance: 2 cm – 450 cm
- Input trigger signal: 10 µs TTL
- Sensor angle: not more than 15 degrees

The sensor modules have the following pins:

Vcc:	+V power
Trig:	Trigger input
Echo:	Echo output
Gnd:	Power ground

Figure 9.37 Ultrasonic transmitter/receiver module

The principle of operation of the ultrasonic sensor module is as follows:

- A 10 µs trigger pulse is sent to the module
- The module then sends eight 40 kHz square wave signals and automatically detects the returned (echoed) pulse signal
- If an echo signal is returned, the time to receive this signal is recorded
- The distance to the object is calculated as:

  Distance to object (in metres) = (time to received echo in seconds * speed of sound) / 2

The speed of sound is 340 m/s, or 0.034 cm/µs

Therefore,

Distance to object (in cm) = (time to received echo in µs) * 0.034 / 2

or,

Distance to object (in cm) = (time to received echo in µs) * 0.017

Figure 9.38 shows the principle of operation of the ultrasonic sensor module. For example, if the time to receive the echo is 294 microseconds, then the distance to the object is calculated as:

Distance to object (cm) = 294 × 0.017 = 5 cm

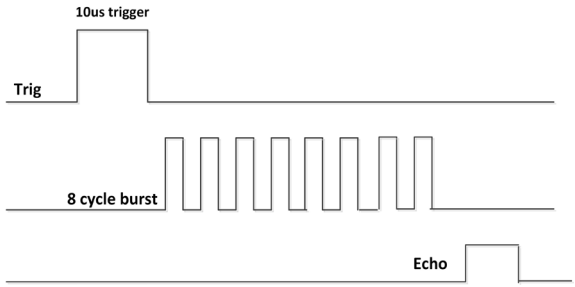

Figure 9.38 Operation of the ultrasonic sensor module

Figure 9.39 shows the circuit diagram of the project. The trig and echo pins of the sensor are connected to GPIO 4 and GPIO 17 respectively. The echo output of the ultrasonic sensor is connected to the Raspberry Pi 5 through a resistive potential divider circuit to drop the voltage level to +3.3 V.

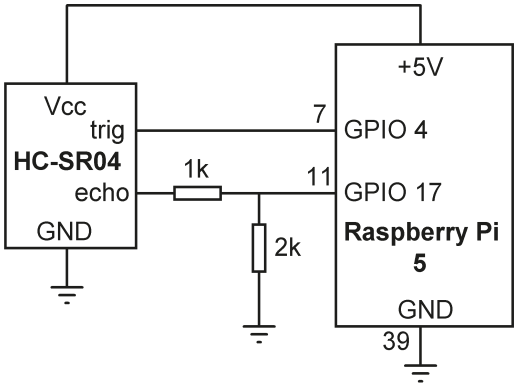

Figure 9.39 Circuit diagram of the project

Program listing: Figure 9.40 shows the program listing (**ultrasonic.py**). At the beginning of the program, module **DistanceSensor** of **gpiozero** is imported to the program. Then the **echo** and **trigger** pins are defined. The remainder of the program runs in a loop where the distance is measured and displayed on the screen. Figure 9.41 shows an example output from the program.

```
#-------------------------------------------------------------
#             ULTRASONIC DISTANCE SENSOR
#             ==========================
#
# This program uses a HC-SR04 type ultrasonic transmitter/receiver
# to measure the distance to an obstacle in-front of the sensor.
# The measured distance is displayed on the screen.
#
# Program: ultrasonic.py
# Date   : October 2023
# Author : Dogan Ibrahim
#-------------------------------------------------------------
from gpiozero import DistanceSensor
from time import sleep

sensor = DistanceSensor(echo=17, trigger=4)

while True:
  print("Distanc (cm)= %6.2f" %(sensor.distance * 100))
  sleep(1)
```

Figure 9.40 Program listing

```
Distanc (cm)=  60.94
Distanc (cm)=  60.94
Distanc (cm)=  26.01
Distanc (cm)=  10.75
Distanc (cm)=   6.23
Distanc (cm)=   5.97
Distanc (cm)=  10.27
Distanc (cm)=  10.85
Distanc (cm)=  12.46
Distanc (cm)=  19.67
Distanc (cm)=   8.42
Distanc (cm)=   8.35
Distanc (cm)=   8.25
```

Figure 9.41 Example output

9.15 Project 12 – Car parking sensors

Description: This is a parking sensors project to help a person park a car safely and easily. A pair of ultrasonic transmitter/receiver sensors is mounted in the front and back of a vehicle to sense the distance to the objects, and an active buzzer sounds if the sensors are too close to the objects in front of them. In this project, safe distance is assumed to be 10 cm.

Block Diagram: Figure 9.42 shows the block diagram of the project.

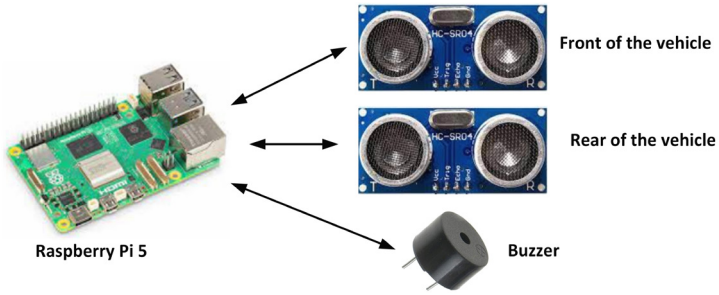

Figure 9.42 Block diagram of the project

Circuit Diagram: Figure 9.43 shows the circuit diagram. The **trig** and **echo** pins of the Front ultrasonic sensor are connected to GPIO 4 and GPIO 17 respectively, as in the previous project. Similarly, the **trig** and **echo** pins of the rear ultrasonic sensor are connected to GPIO 27 and GPIO 22 respectively. Echo outputs of the ultrasonic sensors are connected to the Raspberry Pi 5 through resistive potential divider resistors to drop the voltage levels to +3.3 V. The active buzzer is connected to GPIO 10 of the Raspberry Pi 5.

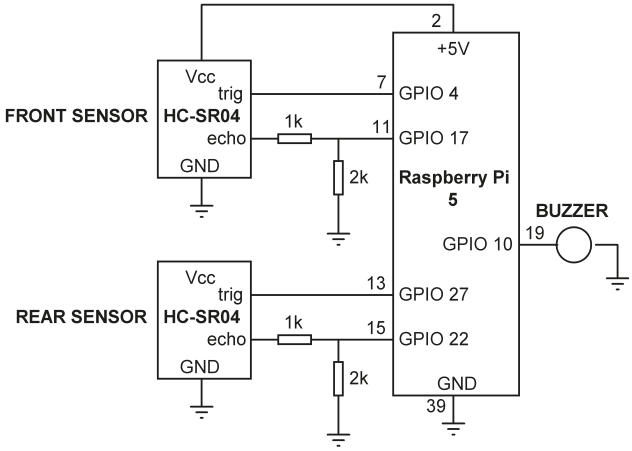

Figure 9.43 Circuit diagram of the project

Program listing: Figure 9.44 shows the program listing (program **parking.py**). Module **DistanceSensor** of **gpiozero** is imported to the program. If the distance of either sensor to the objects is less than or equal to the **Allowed_Distance** (which is set to 10 cm) then the buzzer is sounded to indicate that the vehicle is too close to objects (either at the front or at the rear).

Since the parking sensor is to be operated away from a PC, it is necessary to auto start the program when power is applied to the Raspberry Pi 5. The program name **parking.py** must be included in file **/etc/rc.local** in the following format so that the program starts as soon as the Raspberry Pi 5 starts after a power-up or after a reboot:

python /home/pi/robot2.py &

```
#-------------------------------------------------------------------------
#                          PARKING SENSORS
#                          ===============
#
# This is a parking sensors project. Ultrasonic tranamitter/receiver
# sensors are attached to the front and rear of a vehicle. In addition
# an active buzzer is connected to the Raspberry Oi 5. The program senses
# the objects in the front and rear of the vehicle and sounds the buzzer
# if the vehicle is too close to the objects. In this project a distance
# less than 10cm is considered to be too close.
#
# File  : parking.py
# Date  : October, 2023
# Author: Dogan Ibrahim
#-------------------------------------------------------------------------
from gpiozero import DistanceSensor, LED
from time import sleep

Buzzer = LED(10)                                        # Buzzer at GPIO 10

sensorForward = DistanceSensor(echo=17, trigger=4)
sensorRear = DistanceSensor(echo=22, trigger=27)
Allowed_Distance = 10

Buzzer.off()

while True:
  obstacle_f = sensorForward.distance * 100             # Forward distance
  obstacle_r = sensorRear.distance * 100                # Rear distance
  if obstacle_f <= Allowed_Distance or obstacle_r <= Allowed_Distance:
     Buzzer.on()
  else:
     Buzzer.off()
```

Figure 9.44 Program listing

After applying power, wait until the Raspberry Pi 5 boots, and the program should start automatically. You should remove your Python program name from file **/etc/rc.local** after testing and completing your project so that the program does not start every time you restart your Raspberry Pi 5!

9.16 Project 13 – Fading LED

Description: In this project, an LED is connected to GPIO 21 (pin 40) of the Raspberry Pi 5 through a 470 Ohm current limiting resistor. The brightness of the LED fades in and fades out every second.

Program listing: Figure 9.45 shows the program listing (**FadeLED.py**). Function **pulse()** of **gpoizero** is used to control the LED. The fade-in and fade-out times are set to 1 second each.

```
#------------------------------------------------------
#
#                      FADING LED
#                      ==========
# In this program an LED is connected to the Raspberry Pi 5.
# The brightness of the LED fades by using th PWMLED function
# of gpiozero
#
# Program: FadeLED.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#------------------------------------------------------
from gpiozero import PWMLED

led = PWMLED(21)                       # LED at GPIO 21

#
# Fade-in time and fade-out time are set to 1 second each
#
led.pulse(fade_in_time=1,fade_out_time=1)

while True:
  pass
```

Figure 9.45 Program listing

9.17 Project 14 – Melody maker

Description: This project shows how tones with different frequencies can be generated and sent to a passive buzzer device. The project shows how the simple melody **Happy Birthday** can be played on the buzzer.

Block diagram: The block diagram of the project is shown in Figure 9.46.

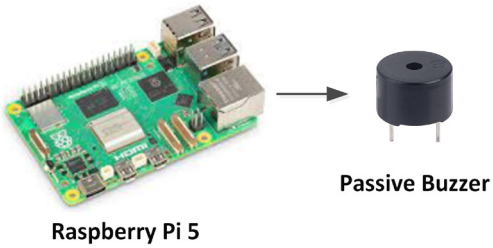

Figure 9.46 Block diagram of the project

Circuit diagram: Figure 9.47 shows the circuit diagram of the project. A passive buzzer is connected to port GPIO 21 (pin 40) of the Raspberry Pi 5. A transistor switch can be used to increase the voltage level of the buzzer (this can be omitted, and the buzzer can be directly connected to GPIO 21 if desired). Any NPN bipolar transistor can be used in this project. The + terminal of the buzzer can be connected to either +3.3 V or to +5V for higher output from the buzzer.

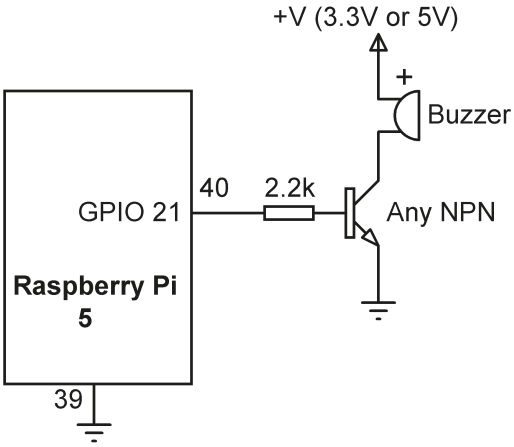

Figure 9.47 Circuit diagram of the project

Melodies

When playing a melody, each note is played for a certain duration and with a certain frequency. In addition, a certain gap is necessary between two successive notes. The frequencies of the musical notes starting from middle C (i.e. C4) are given below. The harmonic of a note is obtained by doubling the frequency. For example, the frequency of C5 is 2 × 262 = 524 Hz.

Notes	C4	C4#	D4	D4#	E4	F4	F4#	G4	G4#	A4	A4#	B4
Hz	261.63	277.18	293.66	311.13	329.63	349.23	370	392	415.3	440	466.16	493.88

To play the tune of a melody, you need to know its musical notes. Each note is played for a certain duration and there is a certain time gap between two successive notes. The next thing we want is to know how to generate a sound with a required frequency and duration. In this project, we will be generating the classic **Happy Birthday** melody, and thus you need to know the notes and their durations. These are given in the table below, where the durations are in units of 400 milliseconds (i.e. the values given in the table should be multiplied by 400 to give the actual durations in milliseconds).

Note	C4	C4	D4	C4	F4	E4	C4	C4	D4	C4	G4	F4	C4	C4	C5	A4	F4	E4	D4	A4	B4	A4	F4	G4	F4
Duration	1	1	2	2	2	3	1	1	2	2	2	3	1	1	2	2	2	2	2	1	1	2	2	2	4

Program Listing: The program listing (program: **Melody**) is shown in Figure 9.48. The notes and their durations are stored in two lists called **Notes** and **Duration,** respectively. Before the main program loop, the durations of each tone are calculated and stored in the array **Duration** so that the main program loop does not have to spend any time doing

these calculations. Inside the program loop, the melody notes are generated with the required durations. A small delay (100 ms) is introduced between each tone. The melody is repeated after five seconds of delay. You can try higher notes for clearer sound, and use speakers instead of a buzzer.

```
#-----------------------------------------------------------
#             PLAY A MELODY (Happy Birthday)
#             ==============================
#
# This program plays the melody Happy Birthday through a buzzer
#
# Program: melody.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#-----------------------------------------------------------
from gpiozero import TonalBuzzer
from gpiozero.tones import Tone
from time import sleep

t = TonalBuzzer(21)

Notes = ['C4','C4','D4','C4','F4','E4','C4','C4','D4','C4','G4',
'F4','C4','C4','C5','A4','F4','E4','D4','A4','B4','A4','F4','G4','F4']

Duration = [1,1,2,2,2,3,1,1,2,2,2,3,1,1,2,2,2,2,2,1,1,2,2,2,4]

length = len(Notes)

while True:
   for i in range(length):
      t.play(Notes[i])
      sleep(Duration[i] * 0.4)
      sleep(0.1)
   t.stop()
   sleep(5)
```

Figure 9.48 Program listing

Chapter 10 • Plotting Graphs with Python and Raspberry Pi 5

10.1 Overview

In this chapter, you will be learning how to draw graphs using the Python programming language. In addition, examples and projects are given on drawing graphs for simple electronic circuits.

10.2 The Matplotlib graph plotting library

Matplotlib is a Python plotting library that is used to create two-dimensional graphs. Before using this package, it has to be installed on your Raspberry Pi 5 using the following command:

```
pi@raspberrypi:~ $ sudo apt-get install python3-matplotlib
```

You must import module matplotlib at the beginning of our programs before we can use Matplotlib using the statement:

```
import matplotlib.pyplot as plt
```

Perhaps the easiest way to learn how to use Matplotlib is to look at an example.

Notice that graphs can only be plotted in Desktop mode. If you are not using a directly connected monitor, then you should start the VNC server on your Raspberry Pi and then start the VNCViewer on your PC to get into the Desktop mode.

Example 1

Write a program to draw a line graph passing from the following (x, y) points:

x: 2 4 6 8
y: 4 8 12 16

Solution 1

The required program listing is shown in Figure 10.1 (program: **graph1.py**). This program is simple. Function call **plt.plot** plots the graph with the specified x and y values. The graph is shown on Desktop when statement **plt.show()** is executed. Start the program by entering the following command in the **Accessories** → **Terminal** windows at the Desktop:

```
pi@raspberrypi:~ $ python graph1.py
```

```
#-----------------------------------------------
#              SIMPLE LINE GRAPH
#              =================
#
# This program draws a line graph passing from
# the following points:
```

```
#
# x = 2 4 6  8
# y = 4 8 12 16
#
# Author: Dogan Ibrahim
# File  : graph1.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt

x = [2, 4, 6, 8]
y = [4, 8, 12, 16]

plt.plot(x, y)
plt.show()
```

Figure 10.1 Program listing

Figure 10.2 shows the graph plotted by the program. Notice that at the bottom of the graph we have several buttons to control the graph, such as zoom, save, etc.

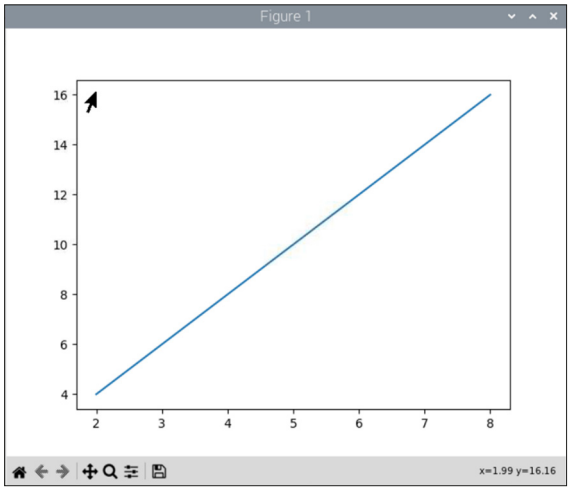

Figure 10.2 Line graph drawn by the program

You can add titles, axis labels, and grid to our graph using the following functions:

```
plt.xlabel(«X values»)
plt.ylabel(«Y values»)
plt.title("Simple X-Y Graph")
plt.grid(True)
```

The new graph is shown in Figure 10.3.

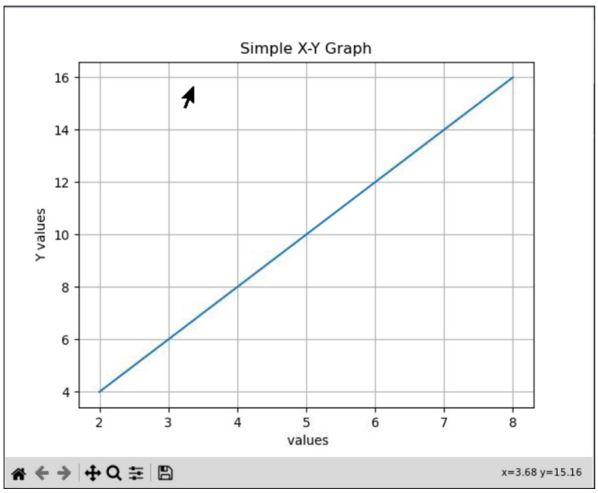

Figure 10.3 Graph with labels, title, and grid

Matplotlib supports a large number of functions (see web link: https://matplotlib.org/stable/api/pyplot_summary.html for a full description of all the functions). Some commonly used functions are:

- bar: make a bar plot
- box: turn the axis box on or off
- boxplot: make a box plot
- figtext: add text to the figure
- hist: plot a histogram
- legend: place a legend on the axes
- loglog: make a logarithmic plot
- pie: plot a pie chart
- polar: make a polar plot
- plotfile: plot data in a file
- semilogx: logarithmic plot with log on x-axis
- semilogy: logarithmic plot with log on y-axis
- suptitle: add a centered title to the plot
- tick_params: change the appearance of ticks and tick labels

Example 2
Write a program to draw a sine curve from 0 to 2π.

Solution 2
You have to use **numpy** arrays to store our data points before plotting. Figure 10.4 shows the program listing (program: **graph2.py**).

```
#-----------------------------------------------
#              SINE GRAPH
#              ==========
#
# This program draws a sine graph from 0 to 2pi
#
# Author: Dogan Ibrahim
# File  : graph2.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np

#
# Calculate the data points in np
#
x = np.arange(0, 2 * np.pi, 0.1)
y = np.sin(x)

#
# Now plot the graph
#
plt.plot(x, y)
plt.xlabel("X values")
plt.ylabel("Sin(X)")
plt.title("Sine Wave")
plt.grid(True)
plt.show()
```

Figure 10.4 Program listing

The graph drawn by the program is shown in Figure 10.5.

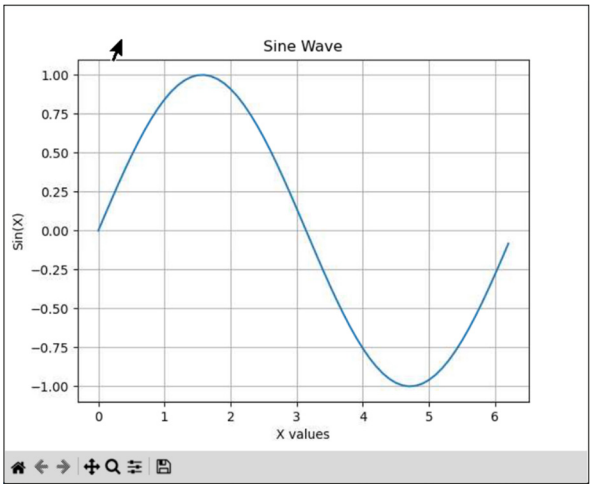

Figure 10.5 Graph drawn by the program

Example 3
Draw the graph of the following function as x is varied from 0 to 4:

$$y = 2x^2 + 3x + 2$$

Solution 3
Figure 10.6 shows the program listing (program: **graph3.py**). After calculating the x and y values, the graph is drawn as shown in Figure 10.7.

```
#-----------------------------------------------
#              Function Graph
#              ==============
#
# This program draws a graph of the function:
#
#    y = 2x2 + 3x + 2 from x=0 to x = 4
#
# Author: Dogan Ibrahim
# File  : graph3.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np

#
# Calculate the data points in np
#
x = np.arange(0, 4, 0.1)
y = [(2 * i * i + 3 * i + 2) for i in x]
```

```
#
# Now plot the graph
#
plt.plot(x, y)
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("y=2x2 + 3x + 2")
plt.grid(True)
plt.show()
```

Figure 10.6 Program listing

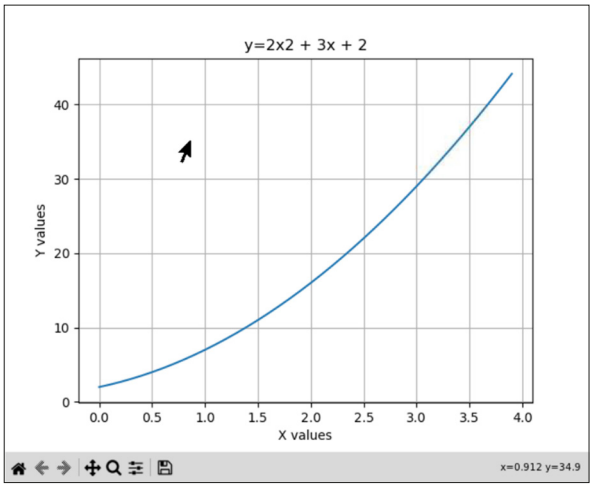

Figure 10.7 Graph drawn by the program

Example 4

This is an example of drawing two graphs on the same axes. Write a program to draw the graphs of the following two functions as x is varied from 0 to 3:

y = x2 + 2
y = x2 + 4

Solution 4

Figure 10.8 shows the program listing (program: **graph4.py**). After calculating the x and y values, the graphs are drawn as shown in Figure 10.9.

```
#-----------------------------------------------
#              Function Graph
#              ==============
#
# This program draws a graph of the functions:
#
#   y = x2 + 2
#   y = x2 + 4 from x=0 to x = 3
#
# Author: Dogan Ibrahim
# File  : graph4.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np

#
# Calculate the data points in np
#
x = np.arange(0, 3, 0.1)
y1 = [(i * i + 2) for i in x]
y2 = [(i * i + 4) for i in x]

#
# Now plot the graph
#
plt.plot(x, y1, linestyle='solid')
plt.plot(x, y2, linestyle='dashed')
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("y=x2+2 and y=x2+4")
plt.grid(True)
plt.show()
```

Figure 10.8 Program listing

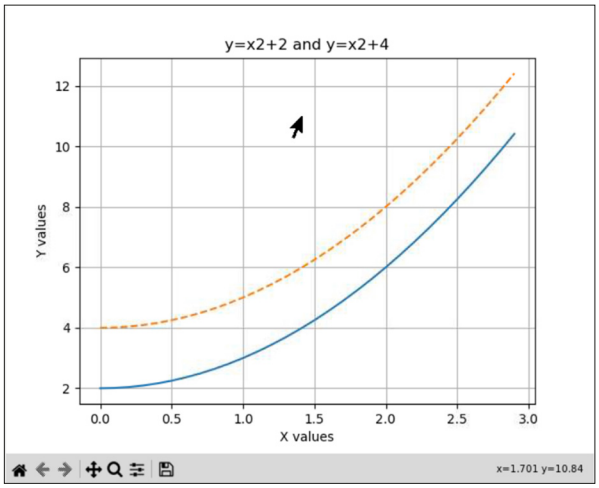

Figure 10.9 Graph drawn by the program

To identify the individual graphs in a multi-graph drawing, you can plot each graph with a different colour, or with different types of lines. Some examples are shown below:

```
plt.plot(x, y1, color='blue')
plt.plot(x, y2, color='green')
```

or

```
plt.plot(x, y1, linestyle='solid')
plt.plot(x, y2, linestyle='dashed')
```

Figure 10.10 shows the graph in Figure 10.9 drawn with different line styles.

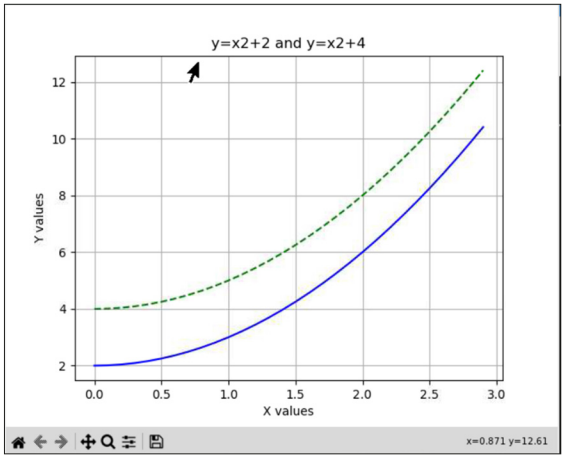

Figure 10.10 Using different line styles

Example 5

In this example, you will use legends to identify multiple graphs in a multigraph drawing. The functions to be drawn are the same as the ones given in the previous example.

Solution 5

Figure 10.11 shows the program listing (program: **graph5.py**). The Matplotlib function **label** is used to identify the two graphs. Also, statement **plt.legend()** must be specified to draw the legend.

```
#-----------------------------------------------
#              Function Graph
#              ==============
#
# This program draws a graph of the functions:
#
#   y = x2 + 2
#   y = x2 + 4 from x=0 to x = 3
#
# In this program the graphs are identified
#
# Author: Dogan Ibrahim
# File  : graph5.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np

#
# Calculate the data points in np
#
x = np.arange(0, 3, 0.1)
y1 = [(i * i + 2) for i in x]
y2 = [(i * i + 4) for i in x]

#
# Now plot the graph
#
plt.plot(x, y1, linestyle='solid', label='x2+2')
plt.plot(x, y2, linestyle='dashed', label='x2+4')
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("y=x2+2 and y=x2+4")
plt.grid(True)
plt.legend()
plt.show()
```

Figure 10.11 Program listing

Figure 10.12 shows the graph drawn by the program.

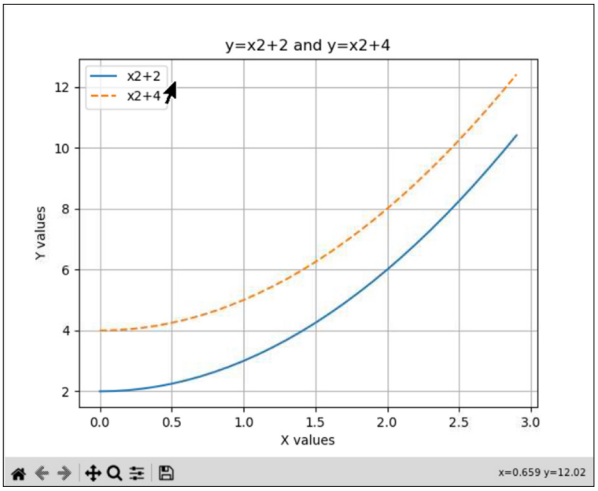

Figure 10.12 Graph drawn by the program

Example 6
Write a program to draw a pie chart for the following data;

France = 15%, Germany = 20%, Italy = 20%, UK = 45%

Solution 6
Figure 10.13 shows the program listing (program: **graph6.py**). The Pie chart is drawn with equal aspect ratio, so that is a circle.

```
#-----------------------------------------------
#              Pie Chart
#              =========
#
# This program draws a pie chart for the data:
#
#   France=15%, Germany=20%,Italy=20%,UK=45%
#
# Author: Dogan Ibrahim
# File  : graph6.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np

labels = "France", "Germany", "Italy", "UK"
sizes = [15, 20, 20, 45]
```

```
x, chrt = plt.subplots()
chrt.pie(sizes, labels=labels)
chrt.axis('equal')
plt.show()
```

Figure 10.13 Program listing

The Pie chart drawn by the program is shown in Figure 10.14.

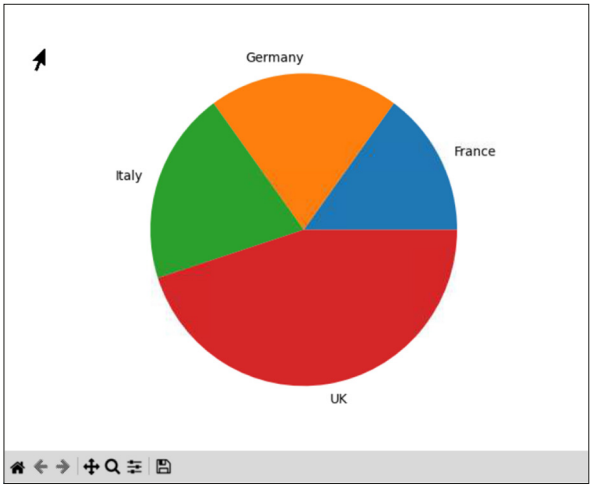

Figure 10.14 Pie chart drawn by the program

We can explode parts of the Pie chart by specifying the parts to be exploded. For example, to explode the fourth item in our example, we can issue the statement:

```
        Explode = (0, 0, 0, 0.1)        # specify amount to be exploded
```

The amount of explosion is determined by the value we specify. Also, the percentages of each part can be written inside the Pie chart elements by using the statement:

```
        autopct='%1.1f%%'               # Specify 1 digit after the decimal point
```

Parts of the Pie chart can be shadowed if desired to give it a 3D effect. This can be done using the statement:

```
        shadow=True
```

The program shown in Figure 10.15 (program: **graph7.py**) makes use of the above features, and the resulting Pie chart is shown in Figure 10.16.

```
#-----------------------------------------------
#                     Pie Chart
#                     =========
#
# This program draws a pie chart for the data:
#
#   France=15%, Germany=20%,Italy=20%,UK=45%
#
# Part UK is exploded in this graph.Also, the
# percentage of each part is written inside the
# corresponding parts and pats are shadowed
#
# Author: Dogan Ibrahim
# File  : graph7.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np

labels = "France", "Germany", "Italy", "UK"
sizes = [15, 20, 20, 45]
explode = (0, 0, 0, 0.1)

x, chrt = plt.subplots()
chrt.pie(sizes, labels=labels, explode=explode,\
autopct='%1.1f%%',shadow=True)
chrt.axis('equal')
plt.show()
```

Figure 10.15 Program listing

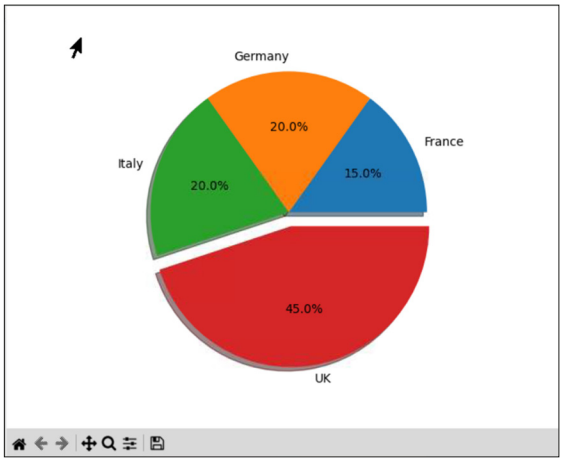

Figure 10.16 Pie chart drawn by the program

Example 7
Write a program to draw a bar chart for the following data:

France = 10, Italy = 8, Germany = 6, UK = 2

Solution 7
Figure 10.17 shows the program listing (program: **graph8.py**). After specifying the values for each bar, the bar chart is drawn.

```
#-----------------------------------------------
#              Bar Chart
#              =========
#
# This program draws a bar chart for the data:
#   France=10, Italy=8,Germany=6,UK=2
#
# Author: Dogan Ibrahim
# File  : graph8.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np

labels = ("France", "Germany", "Italy", "UK")
pos = np.arange(len(labels))
values = [10, 8, 6, 2]

plt.bar(pos, values, align='center',alpha=0.5)
plt.xticks(pos, labels)
plt.ylabel('MB/s')
plt.title('Internet Speed')
plt.show()
```

Figure 10.17 Program listing

Figure 10.18 shows the graph drawn by the program.

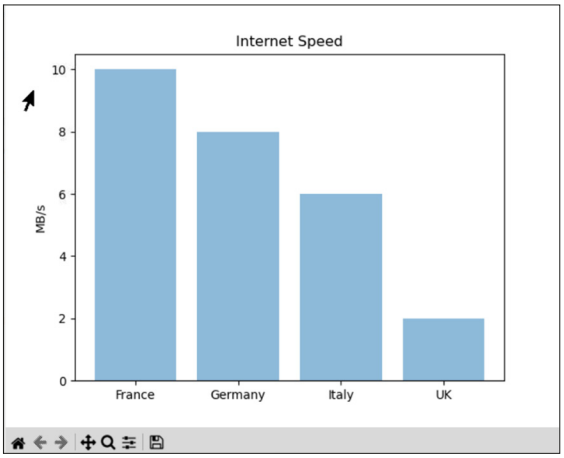

Figure 10.18 Graph drawn by the program

You can plot a horizontal bar chart by replacing the statement **plt.bar** with **plt.barh**.

10.3 Project 1 – RC transient circuit analysis - Charging

Description: This project is about analysing a charging RC transient circuit by plotting its time response.

Background Information: RC circuits are used in many radio and communications circuits. A typical RC transient circuit consists of a resistor in series with a capacitor, as shown in Figure 10.19. When the switch is closed, the voltage across the capacitor rises exponentially with a time constant, T = RC.

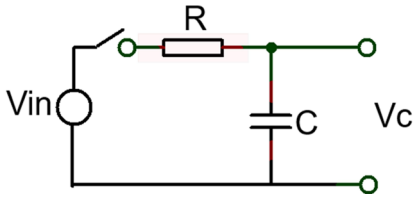

Figure 10.19 Charging RC circuit

Expressed mathematically, assuming that initially the capacitor is discharged, when the switch is closed the voltage across the capacitor rises a given by the following formula:

$$Vc = Vin\left(1 - e^{-t/RC}\right) \qquad (10.1)$$

Initially, the voltage across the capacitor is 0 V, and in steady state the voltage across the capacitor becomes equal to Vin. The time constant is the time when the output voltage rises to around 63.2% of its final value.

Program Listing: Figure 10.20 shows the program listing (program: RCrise.py). After displaying the heading, the values of the input voltage Vin, and resistor and capacitor values

are read from the keyboard. The program then calculates the time constant as T=RC and displays the time constant and also draws the time response of the circuit. The graph is drawn as the time value (x-axis) changes from 0 to 6T, and 50 points are taken to draw the graph. The time constant is also written on the graph at the point (Time constant, Vin/2). The horizontal axis is in seconds, while the vertical axis is in volts.

```
#-----------------------------------------------
#              RC TRANSIENT RESPONSE
#              =====================
#
# This program reads the R and C values and then
# calculates and displays the time conctant. Also,
# the time response of teh circuit is drawn
#
# Author: Dogan Ibrahim
# File  : RCrise.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np
import math

print("RC Transient Response")
print("=====================")

#
# Read Vin, R and C
#
Vin = float(input("Enter Vin in Volts: "))
R = float(input("Enter R in Ohms: "))
C = float(input("Enter C in microfarads: "))
C = C / 1000000.0

#
# Calculate and display time constant
#
T = R * C
F = 6.0 * T
N = F / 50.0
print("Time constant = %f seconds" %(T))

#
# Now plot the time response
#
x = np.arange(0, F, N)
y = [(Vin * (1.0 - math.exp(-i/T))) for i in x]
```

```
plt.plot(x, y)
plt.xlabel("Time (s)")
plt.ylabel("Capacitor Volts")
plt.title("RC Response")
plt.grid(True)
TC = "T="+str(T)+"s"
plt.text(T, Vin/2, TC)
plt.show()
```

Figure 10.20 Program listing

Figure 10.21 shows an example graph displayed by the program. In this program, the following input values were used (see Figure 10.22):

Vin = 10 V
R = 100 Ω
C = 10 µF

The time constant was calculated to be 0.1 seconds.

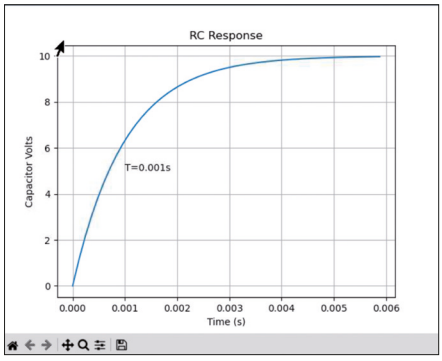

Figure 10.21 The graph plotted by the program

```
pi@raspberrypi:~ $ python RCrise.py
RC Transient Response
=====================
Enter Vin in Volts: 10
Enter R in Ohms: 100
Enter C in microfarads: 10
Time constant = 0.001000 seconds
```

Figure 10.22 Input values to the example program

10.4 Project 2 – RC transient circuit analysis - Discharging

Description: This case study is about analysing a discharging RC transient circuit by plotting its time response.

Background Information: In this case study an RC circuit is used as in Figure 10.23. We assume that the capacitor is fully charged after switch s1 is closed. We then close switch s2 so that the capacitor discharges through resistor R. The time response of the voltage across the capacitor is then given by:

$$Vc = Vo\ e^{-t/RC} \tag{10.2}$$

Where Vo is the initial voltage across the capacitor (normally same as Vin) before s2 is closed. Again, T=RC is known as the time constant of the circuit.

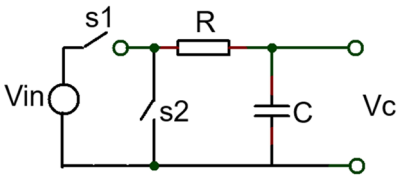

Figure 10.23 Discharging RC circuit

Program Listing: Figure 10.24 shows the program listing (program: RCfall.py). After displaying the heading, the values of the initial voltage across the capacitor (Vo), the resistor and the capacitor are read from the keyboard. The program then calculates the time constant as T=RC and displays the time constant and draws the time response of the circuit. The graph is drawn as the time value (x-axis) changes from 0 to 6T and 50 points are taken to draw the graph. The time constant is also written on the graph at the point (Time constant, Vo/2). The horizontal axis is in seconds, while the vertical axis is in volts.

```
#-----------------------------------------------
#              RC TRANSIENT RESPONSE
#              =====================
#
# This program reads the R and C values and then
# calculates and displays the time constant. Also,
# the time response of the circuit is drawn as the
# capacitor is discharged
#
# Author: Dogan Ibrahim
# File  : RCfall.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np
import math

print("RC Transient Response")
print("=====================")
```

```
#
# Read Vo, R and C
#
Vo = float(input("Enter Initial Capacitor Voltage in Volts: "))
R = float(input("Enter R in Ohms: "))
C = float(input("Enter C in microfarads: "))
C = C / 1000000.0

#
# Calculate and display time constant
#
T = R * C
F = 6.0 * T
N = F / 50.0
print("Time constant = %f seconds" %(T))

#
# Now plot the time response
#
x = np.arange(0, F, N)
y = [(Vo * (math.exp(-i/T))) for i in x]

plt.plot(x, y)
plt.xlabel("Time (s)")
plt.ylabel("Capacitor Volts")
plt.title("RC Response")
plt.grid(True)
TC = "T="+str(T)+"s"
plt.text(T, Vo/2, TC)
plt.show()
```

Figure 10.24 Program listing

Figure 10.25 shows an example graph displayed by the program. In this program, the following input values were used (see Figure 10.26):

Initial capacitor voltage = 10 V
R = 1000 Ω
C = 100 μF

The time constant was calculated to be 0.1 seconds.

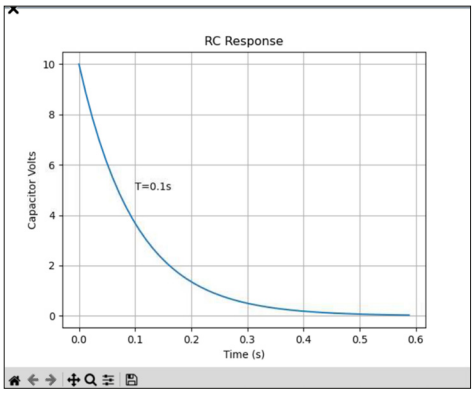

Figure 10.25 The graph plotted by the program

```
pi@raspberrypi:~ $ python RCfall.py
RC Transient Response
=====================
Enter Initial Capacitor Voltage in Volts: 10
Enter R in Ohms: 1000
Enter C in microfarads: 100
Time constant = 0.100000 seconds
```

Figure 10.26 Input values to the example program

10.5 Transient RL circuits

The time response of a transient resistor-inductor circuit is similar to the RC circuit. When the circuit is connected to a DC supply of value Vin, the current in the circuit rises exponentially and is given by the following formula:

$$i = \frac{Vin}{R}\left(1 - e^{-Rt/L}\right) \tag{10.3}$$

Where, Vin is in volts, R in ohms, L in henries, and t in seconds. The time constant of this circuit is given by T = L/R.

After the current reaches its steady state value, disconnecting the DC supply and shorting the leads causes the current in the circuit to fall exponentially, given by the following formula:

$$i = \frac{Vo}{R}e^{-Rt/L} \tag{10.4}$$

Where, Vo is the initial voltage across the inductor.

The transient response of RL circuits is similar to those of the RC circuits and therefore are not covered further in this book.

10.6 Project 3 – RCL transient circuit analysis

Description: This case study is about analysing the transient response of a second order series connected RLC circuit by plotting its time response.

Background Information: An RLC circuit (Figure 10.27) is a second-order system that can have 3 modes of operation depending on the values of the components when a DC voltage is applied across its terminals.

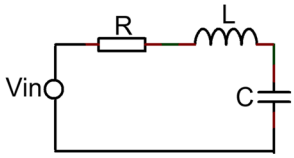

Figure 10.27 RLC circuit

Underdamped mode: This mode is identified when the following condition holds true:

$$R < 2\sqrt{\frac{L}{C}} \tag{10.5}$$

When DC voltage is applied to the circuit, the current in the circuit is given by the following formula:

$$i(t) = \frac{Vin}{WL\sqrt{1-\xi^2}} e^{-\xi Wt} sin\left(Wt\sqrt{1-\xi^2}\right) \tag{10.6}$$

Where,

$$W = \frac{1}{\sqrt{LC}} \text{ and, } \xi = \frac{R}{2}\sqrt{\frac{C}{L}} < 1 \tag{10.7}$$

Critically damped mode: In this mode of operation, the following is satisfied:

$$R = 2\sqrt{\frac{L}{C}} \tag{10.8}$$

When DC voltage is applied to the circuit, the current in the circuit is given by the following formula:

$$i(t) = \frac{Vin}{L} te^{-Wt} \tag{10.9}$$

Where,

$$W = \frac{1}{\sqrt{LC}} \text{ and, } \xi = \frac{R}{2}\sqrt{\frac{C}{L}} = 1 \tag{10.10}$$

Overdamped mode: In this mode of operation, the following is satisfied:

$$R > 2\sqrt{\frac{L}{C}} \tag{10.11}$$

When DC voltage is applied to the circuit, the current in the circuit is given by the following formula:

$$i(t) = \frac{Vin}{WL\sqrt{\xi^2-1}} e^{-\xi Wt} sinh\left(Wt\sqrt{\xi^2-1}\right) \quad (10.12)$$

Where,

$$W = \frac{1}{\sqrt{LC}} \text{ and, } \xi = \frac{R}{2}\sqrt{\frac{C}{L}} > 1 \quad (10.13)$$

Program Listing: Figure 10.28 shows the program listing (program: **RLC.py**). At the beginning of the program a heading is displayed and then the values of the input voltage, resistor, capacitor and the inductor are read and stored in variables Vin, R, C, and L respectively. The program then finds out in which mode the circuit will be operating based on the value of ξ. Then, three functions are used, one for each mode, to calculate and plot the transient response of the circuit. The mode of the circuit is displayed on the graph at the coordinate (3T, 0), where T = 2π/W. In all the graphs, 80 points are used to draw the points from 0 to 6T.

```
#-----------------------------------------------
#              RLC TRANSIENT RESPONSE
#              ======================
#
# This program reads the R,L,C values and then
# calculates and displays the transient response
#
# Author: Dogan Ibrahim
# File  : RLC.py
# Date  : October, 2023
#-----------------------------------------------
import matplotlib.pyplot as plt
import numpy as np
import math
global x, y, z

def critically_damped():
  global x,y
  x = np.arange(0, F, N)
  y = [Vin*((1/L) * i * math.exp(-i*w)) for i in x]

def underdamped():
  global x,y,z
  x = np.arange(0, F, N)
  zeta = math.sqrt(1 - z*z)
  y = [Vin*(1/(w*L*zeta)*(math.exp(-z*w*i))*math.sin(w*i*zeta)) for i in x]
```

```
def overdamped():
  global x,y,z
  x = np.arange(0, F, N)
  y = [Vin*(1/(w*L*(math.sqrt(z*z-1))))*(math.exp(-z*w*i))*\
math.sinh(w*i*math.sqrt(z*z-1)) for i in x]


print("RLC Transient Response")
print("======================")

#
# Read Vin, R,C and L
#
Vin = float(input("Enter Vin in Volts: "))
R = float(input("Enter R in Ohms: "))
C = float(input("Enter C in microfarads: "))
C = C / 1000000.0
L = float(input("Enter L in millihenries: "))
L = L / 1000.0
w = math.sqrt(1/(L * C))
z = (R/2) * math.sqrt(C / L)
T = (2.0 * math.pi) / w
F = 6 * T
N = F / 80.0

#
# Find the mode of operation
#
mode = R - 2.0 * math.sqrt(L / C)
if abs(mode) < 0.01:
  case = 2
  md = "Critically Damped"
  critically_damped()
elif mode < 0:
  case = 1
  md = "Underdamped"
  underdamped()
elif mode > 0:
  case = 3
  md = "Overdamped"
  overdamped()

#
# Now plot the time response
#
```

```
plt.plot(x, y)
plt.xlabel("Time (s)")
plt.ylabel("Current")
plt.title("RLC Response")
plt.grid(True)
plt.text(3*T,0,  md)
plt.show()
```

Figure 10.28 Program listing

Figure 10.29 shows a typical run of the program with the following values:

Vin = 10 V
R = 10 Ω
C = 100 µF
L = 200 µH

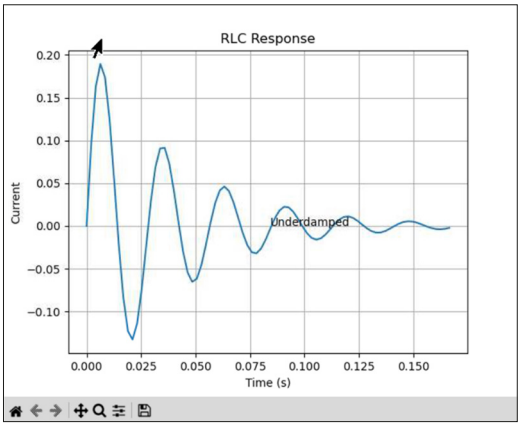

Figure 10.29 Response of the circuit

10.7 Project 4 – Temperature, pressure and humidity measurement – Display on the screen

Description: In this project the BME280 sensor module is used to read the ambient temperature, pressure and humidity and to display the readings on the screen.

Block diagram: Figure 10.30 shows the block diagram of the project.

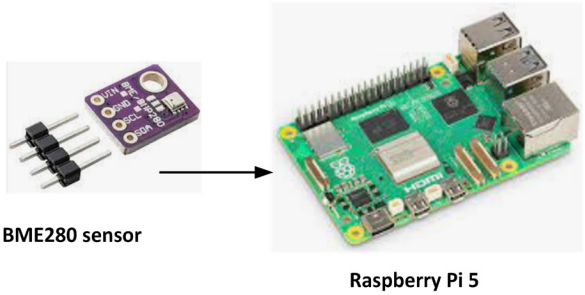

Figure 10.30 Block diagram of the project

The BME280 sensor module

The BME280 module (Figure 10.31) is a low-cost sensor developed for measuring the ambient temperature, atmospheric pressure, and the humidity. This module operates with the I²C (or SPI) bus interface and has the pins SDA, SCL, Vin, and GND. The basic specifications of this module are:

- Operating voltage: 1.2 to 3.6 V
- Interface I²C or SPI
- Current consumption: 1.8 µA
- Humidity sensor response time: 1 s
- Humidity sensor accuracy: ±3%
- Pressure sensor range: 300 to 1100 hPa
- Temperature range: -40 to +85ºC

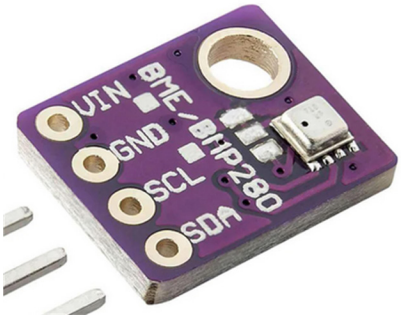

Figure 10.31 The BME280 sensor module

Circuit diagram: The project circuit diagram is shown in Figure 10.32. The module is connected to Raspberry Pi SDA (pin 3) and SCL (pin 5) pins. +3.3 V power is applied from pin 1.

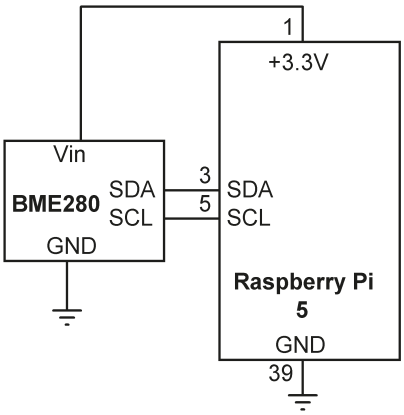

Figure 10.32 Circuit diagram of the project

The default address of the BME280 is 0x76. This can be confirmed by entering the following command after the circuit is built (Figure 10.33):

```
i2cdetect –y 1
```

```
pi@raspberrypi:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 --
pi@raspberrypi:~ $
```

Figure 10.33 Checking the I²C bus for the sensor module

Program listing: Figure 10.34 shows the program listing (**bme280.py**). Before running the program, it is necessary to load the BME280 library. The steps are:

- git clone https://github.com/MarcoAndreaBuchmann/bme280pi.git
- cd bme280pi
- python setup.py install

The sensor library can be imported to your Python programs as follows:

```
from bme280pi import Sensor
sensor = Sensor()
```

At the beginning of the program, the BME280 sensor library is imported s above, Inside the main program loop the temperature, atmospheric pressure, and humidity are read and displayed on the screen every 5 seconds.

```
#-------------------------------------------------------------
#         TEMPERATURE,ATMOSPHERIC PRESSURE AND HUMIDITY
#         ============================================
#
# This program reads the ambient temperature, atmospheric
# pressure, and humidity using a BME280 sensor module. The
# readings are dislayed on the screen every 5 seconds
#
# Program: bme280.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#-------------------------------------------------------------
from time import sleep
from bme280pi import Sensor

sensor = Sensor(address = 0x76)

while True:                                    # infinite loop
  data = sensor.get_data()                     # get sensor data
  temperature = data['temperature']            # temperature
  pressure = data['pressure']                  # pressure
  humidity = data['humidity']                  # humidity
  print("Temperature = %5.2f C" %temperature)
  print("Pressure = %d hPa" %pressure)
  print("Humidity = %d" %humidity)
  print("")
  sleep(5)
```

Figure 10.34 Program listing

Figure 10.35 shows an example output from the program.

```
pi@raspberrypi:~ $ python bme280.py
Temperature = 26.16 C
Pressure = 1004 hPa
Humidity = 60

Temperature = 26.13 C
Pressure = 1004 hPa
Humidity = 60

Temperature = 27.68 C
Pressure = 1004 hPa
Humidity = 62

Temperature = 30.13 C
Pressure = 1004 hPa
Humidity = 61
```

Figure 10.35 Output from the program

10.8 Project 5 – Temperature, pressure and humidity measurement – Plotting the data

Description: This project is very similar to the previous one, but here the data is plotted on the Desktop.

The block diagram and circuit diagram of the project are the same as in Figure 10.30 and Figure 10.32.

Program listing: Figure 10.36 shows the program listing (**bme280plot.py**). The sensor data is collected for 60 seconds where the temperature, pressure and humidity are stored in **t[]**, **p[]**, and **h[]**. The seconds are stored in **tim[]**. When the program runs, the message **Collecting data...** is displayed. The collected data is plotted as shown in Figure 10.37. Note that you can adjust the position of the graphs on the screen using the horizontal arrow tool at the bottom of the screen.

```
#-------------------------------------------------------------
#     PLOT TEMPERATURE,ATMOSPHERIC PRESSURE AND HUMIDITY
#     ================================================
#
# This program reads the ambient temperature, atmospheric
# pressure, and humidity using a BME280 sensor module. The
# readings are plotted on the Desktop
#
# Program: bme280plot.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#-------------------------------------------------------------
from time import sleep
from bme280pi import Sensor
import matplotlib.pyplot as plt

sensor = Sensor(address = 0x76)

p = [0]*60
t=[0]*60
h=[0]*60
data = [0]*60
tim=[0]*60
print("Collecting data...")

for i in range(60):
   data=sensor.get_data()
   tim[i]=i
   p[i] =int(data['pressure'])
   t[i] = int(data['temperature'])
   h[i] = int(data['humidity'])
```

```
   sleep(0.1)

plt.figure()
plt.subplot(2, 2, 1)
plt.plot(tim,t)
plt.title("Temperature (C)")
plt.grid()

plt.subplot(2, 2, 2)
plt.plot(tim,p)
plt.title("Pressure (hPa)")
plt.grid()

plt.subplot(2, 2, 3)
plt.plot(tim,h)
plt.title("Relative Humidity (%)")
plt.grid()
plt.show()
```

Figure 10.36 Program listing

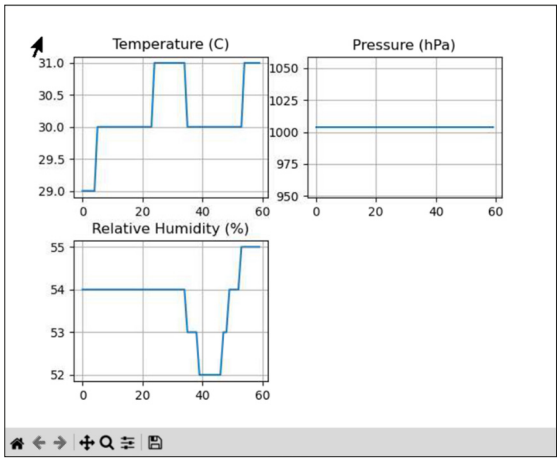

Figure 10.37 Example output from the program

Chapter 11 • Waveform Generation – Using the Digital-to-Analog Converter (DAC)

11.1 Overview

Waveform generators are important in many electronic communication applications. In this chapter, you will be developing projects to generate various waveforms such as square, sine, triangular, staircase, etc. by using an external DAC chip and programming the Raspberry Pi 5. In this book, you will be using the popular MCP4921 DAC chip from Microchip.

11.2 The MCP4921 DAC

Before using the MCP4921, it is worthwhile to look at its features and operation in some detail. MCP4921 is a 12-bit DAC that operates with the SPI bus interface. Figure 11.1 shows the pin layout of this chip. The basic features are:

- 12-bit operation
- 20 MHz clock support
- 4.5 µs settling time
- External voltage reference input
- 1× or 2× gain
- 2.7 to 5.5 V supply voltage
- -40ºC to +125ºC temperature range

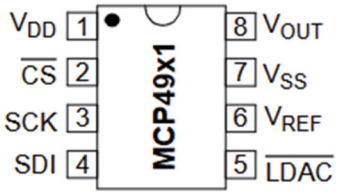

Figure 11.1 MCP4921 DAC

The pin descriptions are:

Vdd:	supply voltage
CS:	chip select (active LOW)
SCK:	SPI clock
SDI:	SPI data in
LDAC:	Used to transfer input register data to the output (active LOW)
Vref:	Reference input voltage
Vout:	analog output
Vss:	supply ground

In projects in this book, you will be operating the MCP4921 with a gain of one. As a result, with a reference voltage of 3.3 V and 12-bit conversion data, the LSB resolution of the DAC will be 3300/4096 = 0.8 mV

The SPI Bus
As it was discussed in an earlier chapter, the Serial Peripheral Interface (SPI) bus consists of two data wires and one clock wire. Additionally, a chip enable (CE or CS) connection is used to select a slave in a multi-slave system. The wires used are:

MOSI (or **SDI)**: **M**aster **O**ut **S**lave **I**n. This signal is output from the master and is input to a slave

MISO: **M**aster **I**n **S**lave **O**ut. This signal is output from a slave and input to a master

SCLK (or **SCK)**: The clock, controlled by the master

CE (r **CS)**: **C**hip **E**nable (slave select)

The following pins are the SPI bus pins on Raspberry Pi 4:

GPIO pin	SPI	Physical pin no
GPIO 10	MOSI (SPI0)	19
GPIO 9	MISO (SPI0)	21
GPIO 11	SCLK (SPI0)	23
GPIO 8	CE0 (SPI0)	24
GPIO 7	CE1 (SPI0)	26
GPIO 20	MOSI (SPI1)	38
GPIO 19	MISO (SPI1)	35
GPIO 21	SCLK (SPI1)	40
GPIO 18	CE0 (SPI1)	12
GPIO 17	CE1 (SPI1)	11

The SPI bus must be enabled using the **raspi-config** console command on the Raspberry Pi before it can be used.

11.3 Project 1 - Generating a square wave signal with any peak voltage up to +3.3 V

Description: In this project you will be using the DAC to generate a square wave signal with the frequency of 1 kHz where the required output voltage is 2 V peak.

Block Diagram: Figure 11.2 shows the block diagram of the project. The output

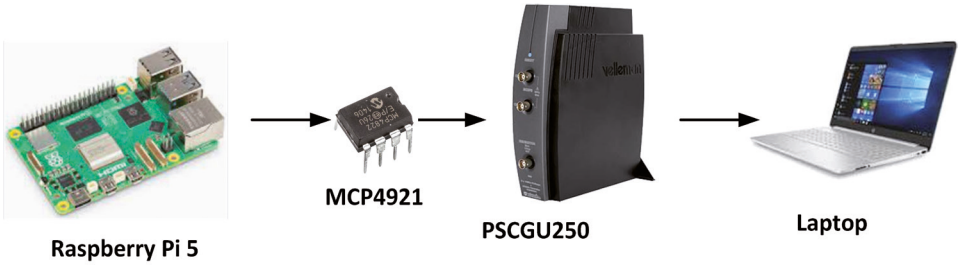

Figure 11.2 Block diagram of the project

Circuit Diagram: The circuit diagram of the project is shown in Figure 11.3. The output of the DAC is connected to a PSCGU250 type digital oscilloscope.

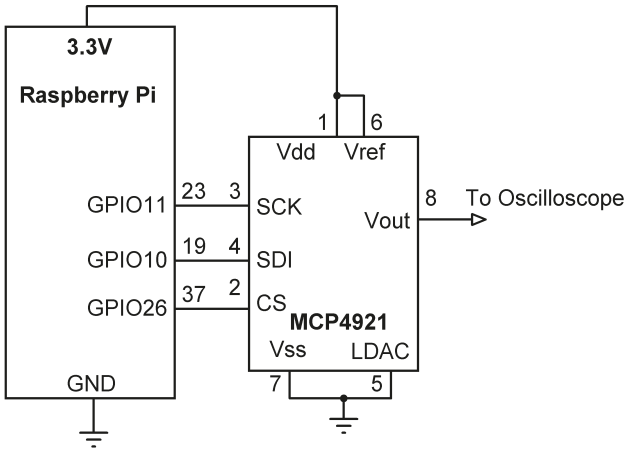

Figure 11.3 Circuit diagram of the project

Program Listing: Data is written to the DAC in 2 bytes. The lower byte specifies D0:D8 of the digital input data. The upper byte consists of the following bits:

D8: D11bits D8:D11 of the digital input data
SHDN 1: active (output available), 0: shutdown the device
GA output gain control. 0: gain is 2x, 1: gain is 1x
BUF 0: input unbuffered, 1: input buffered
A/B 0: write to DACa, 1: Write to DACb (MCP4921 supports only DACa)

In normal operation, we will send the upper byte (D8:D11) of the 12-bit (D0:D11) input data with bits D12 and D13 set to 1 so that the device is active, and the gain is set to 1x. Then we will send the low byte (D0:D7) of the data. This means that 0x30 should be added to the upper byte before sending it to the DAC.

Figure 11.4 shows the program listing (program: **squaredac.py**). GPIO 26 is used as the **CS** pin. The **frequency** variable is set to 1000, which is the required frequency. Function DAC sends the 12-bit input data to the DAC. This function has two parts. In the first part, the HIGH byte is sent after adding 0x30 as described above. Function **xfer2** is used to send the data to the DAC. In the second part of the function, the LOW byte is extracted and is sent to the DAC. Notice that we could have sent both the high byte and the low byte using the same **xfer2** function, as follows:

```
highbyte = (data >> 8) & 0x0F
highbyte = highbyte + 0x30

lowbyte = data & 0xFF
xfer2([highbyte, lowbyte])
```

Variable **ONvalue** is set to 2000 × 4095/3300, which is the digital value corresponding to 2000 mV (i.e. 2 V, remember that the 12-bit DAC has 4095 steps, and the reference voltage is set to 3300 mV). The **OFFvalue** is set to 0 V. Normally, the delay between the ON and OFF times should have been equal to **halfperiod**. However, it was found by experiment that the DAC routine takes about 0.2 ms (0.0002 second) and this changes the period and consequently the frequency of the output waveform. Because of this, 2 mV is subtracted from **halfperiod** as shown in Figure 13.9

```
#-----------------------------------------------------------------
#                GENERATE SQUARE WAVEFORM
#                ========================
#
# This program generates square waveform with the frequency 1kHz.
# In this program the MC4921 DAC chip is used to set the output
# peak voltage to 2V
#
# Author: Dogan Ibrahim
# File  : squaredac.py
# Date  : October, 2023
#-----------------------------------------------------------------
from gpiozero import LED
from time import sleep
import spidev                                   # Import SPI

spi = spidev.SpiDev()
spi.open(0, 0)                                  # Bus=0, device=0
spi.max_speed_hz = 3900000

CS = LED(26)                                    # GPIO26 is CS output
CS.on()                                         # DIsable CS

frequency = 1000                                # Required Frequency
```

```
period = 1 / frequency                          # Period of the signal
halfperiod = period / 2                         # Half period

#
# This function implements the DAC. The data in "data" is sent
# to the DAC
#
def DAC(data):
   CS.off()                                     # Enable CS
#
# Send HIGH byte
#
   temp = (data >> 8) & 0x0F                    # Get upper byte
   temp = temp + 0x30                           # OR with 0x30
   spi.xfer2([temp])                            # Send to DAC
#
# Send LOW byte
#
   temp = data & 0xFF                           # Get lower byte
   spi.xfer2([temp])                            # Send to DAC

   CS.on()                                      # Disable CS

try:
  ONvalue = int(2000*4095/3300)                 # 2V output
  OFFvalue = 0

  while True:
     DAC(ONvalue)                               # Send to DAC
     sleep(halfperiod - 0.0002)                 # Wait
     DAC(OFFvalue)                              # Send to DAC
     sleep(halfperiod - 0.0002)                 # Wait

except KeyboardInterrupt:
     pass
```

Figure 11.4 Program listing

Figure 11.5 shows the output waveform generated by the program. Notice that the peak output voltage is 2 V, as expected.

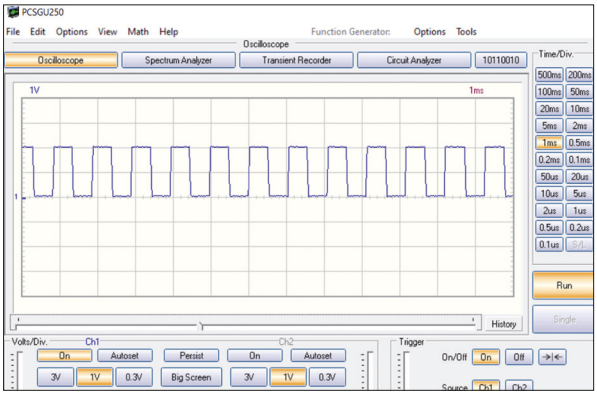

Figure 11.5 Output waveform

11.4 Project 2 – Generating a sawtooth wave signal

Description: In this project, you will be using the DAC to generate a sawtooth wave signal with the following specifications:

Peak voltage:	3.3 V
Step width:	1 ms
Number of steps:	6

The block diagram and circuit diagram of the project are as in Figure 11.2 and Figure 11.3

Program Listing: Figure 11.6 shows the program listing (program: **sawtooth.py**). The program is very similar to the one given in Figure 11.4.

```
#-----------------------------------------------------------------
#                GENERATE SAWTOOTH WAVEFORM
#                ==========================
#
# This program generates sawtooth waveform with 6 steps where each
# step has a width of 1ms
#
# Author: Dogan Ibrahim
# File  : sawtooth.py
# Date  : October, 2023
#-----------------------------------------------------------------
from gpiozero import LED
from time import sleep                          # Import time
import spidev                                   # Import SPI

spi = spidev.SpiDev()
spi.open(0, 0)                                  # Bus=0, device=0
spi.max_speed_hz = 3900000
```

```
CS = LED(26)                                    # GPIO26 is CS output
CS.on()                                         # Disable CS

#
# This function implements the DAC. The data in "data" is sent
# to the DAC
#
def DAC(data):
   CS.off()                                     # Enable CS
#
# Send HIGH byte
#
   temp = (data >> 8) & 0x0F                    # Get upper byte
   temp = temp + 0x30                           # OR with 0x30
   spi.xfer2([temp])                            # Send to DAC
#
# Send LOW byte
#
   temp = data & 0xFF                           # Get lower byte
   spi.xfer2([temp])                            # Send to DAC

   CS.on()                                      # Disable CS

try:
  while True:                                   # Do forever
     i = 0
     while i < 1.1:
        DACValue = int(i*4095)                  # Value  to send
        DAC(DACValue)                           # Send to DAC
        sleep(0.0007)                           # Wait
        i = i + 0.2

except KeyboardInterrupt:
    pass
```

Figure 11.6 Program listing

An example output waveform taken from the oscilloscope is shown in Figure 11.7. Notice that the time delay had to be adjusted experimentally to give the correct timing.

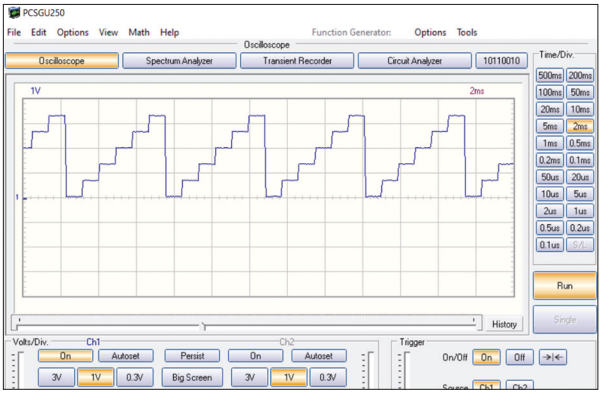

Figure 11.7 Example output waveform

11.5 Project 3 - Generating a triangle wave signal

Description: In this project, we will be using the DAC to generate a triangle wave signal.

The block diagram and circuit diagram of the project are as in Figure 11.2 and Figure 11.3

Program Listing: Figure 11.8 shows the program listing (program: **triangle.py**). The program is very similar to the one given in Figure 11.6.

```
#-----------------------------------------------------------------
#                GENERATE TRIANGLE WAVEFORM
#                ==========================
#
# This program generates triangle waveform
#
# Author: Dogan Ibrahim
# File  : triangle.py
# Date  : October, 2023
#-----------------------------------------------------------------
from gpiozero import LED
from time import sleep                          # Import time
import spidev                                   # Import SPI

spi = spidev.SpiDev()
spi.open(0, 0)                                  # Bus=0, device=0
spi.max_speed_hz = 3900000

CS = LED(26)
CS.on()                                         # Disable CS
sample = 0
Inc = 0.05

#
```

```
# This function implements the DAC. The data in "data" is sent
# to the DAC
#
def DAC(data):
   CS.off()                                  # Enable CS
#
# Send HIGH byte
#
   temp = (data >> 8) & 0x0F                 # Get upper byte
   temp = temp + 0x30                        # OR with 0x30
   spi.xfer2([temp])                         # Send to DAC
#
# Send LOW byte
#
   temp = data & 0xFF                        # Get lower byte
   spi.xfer2([temp])                         # Send to DAC

   CS.on()                                   # Disable CS

try:
   while True:
        DACValue = int(sample*4095)          # Value  to send
        DAC(DACValue)                        # Send to DAC
        sleep(0.0001)                        # Wait
        sample = sample + Inc                # Next sample
        if sample > 1.0 or sample < 0:
           Inc = -Inc
           sample = sample + Inc

except KeyboardInterrupt:
    pass
```

Figure 11.8 Program listing

An example output waveform taken from the oscilloscope is shown in Figure 11.9.

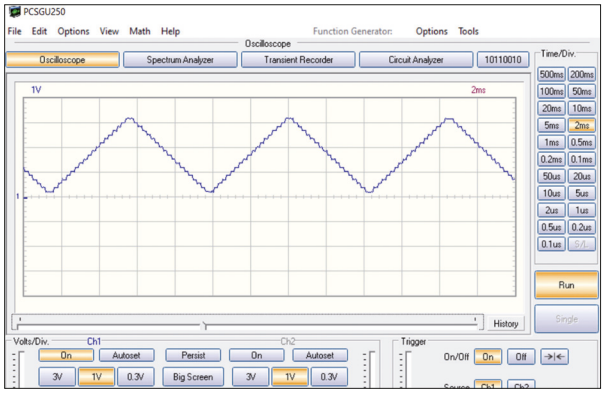

Figure 11.9 Example output waveform

11.6 Project 4 - Generating an arbitrary wave signal

Description: In this project, you will be using the DAC to generate an arbitrary waveform. One period of the shape of the waveform will be sketched and values of the waveform at different points will be extracted and loaded into a lookup table. The program will output the data points at the appropriate times to generate the required waveform.

The shape of one period of the waveform to be generated is shown in Figure 11.10. Notice that the waveform has a period of 20 ms.

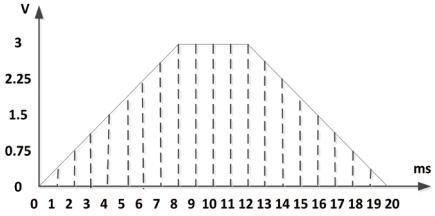

Figure 11.10 Waveform to be generated

The waveform takes the following values:

Time (ms)	Amplitude (V)	Time (ms)	Amplitude (V)
0	0	11	3.00
1	0.375	12	3.00
2	0.75	13	2.625
3	1.125	14	2.25
4	1.50	15	1.875
5	1.875	16	1.50
6	2.25	17	1.125
7	2.625	18	0.75
8	3.00	19	0.375
9	3.00	20	0
10	3.00		

The block diagram and circuit diagram of the project are as in Figure 11.2 and Figure 11.3

Program Listing: Figure 11.11 shows the program listing (program: **arbit.py**). The sample points of the waveform are stored in a list called **wave**. Variable **sample** indexes this list and sends the sample values to the DAC. The time of each sample was specified to be 1 ms. It was found by experiment that a 0.8 ms delay gave the correct results because of the delay in the DAC routine.

```
#-----------------------------------------------------------------
#                GENERATE ARBITRARY WAVEFORM
#                ===========================
#
# This program generates an arbitrary waveform whose sample points
# are defined in the program
#
# Author: Dogan Ibrahim
# File  : arbit.py
# Date  : October, 2023
#-----------------------------------------------------------------
from gpiozero import LED
from time import sleep                          # Import time
import spidev                                   # Import SPI

spi = spidev.SpiDev()
spi.open(0, 0)                                  # Bus=0, device=0
spi.max_speed_hz=3900000

CS = LED(26)                                    # GPIO26 is CS output
CS.on()                                         # Disable CS
sample = 0

#
# Waveform sample points
#
wave = [0,0.375,0.75,1.125,1.5,1.875,2.25,2.625,3,3,3,3,3,\
2.625,2.25,1.875,1.5,1.125,0.75,0.375,0]

#
# This function implements the DAC. The data in "data" is sent
# to the DAC
#
def DAC(data):
   CS.off()                                     # Enable CS
#
# Send HIGH byte
#
```

```
    temp = (data >> 8) & 0x0F                    # Get upper byte
    temp = temp + 0x30                           # OR with 0x30
    spi.xfer2([temp])                            # Send to DAC

    temp = data & 0xFF
    spi.xfer2([temp])

    CS.on()                                      # Disable CS

try:
    while True:
         DACValue = int(wave[sample]*4095/3.3)  # Value  to send
         DAC(DACValue)                           # Send to DAC
         sample = sample + 1                     # Inc sample index
         sleep(0.0008)                           # Wait
         if sample == 20:                        # If 20 sampes
            sample = 0

except KeyboardInterrupt:
    pass
```

Figure 11.11 Program listing

An example output waveform taken from the oscilloscope is shown in Figure 11.12.

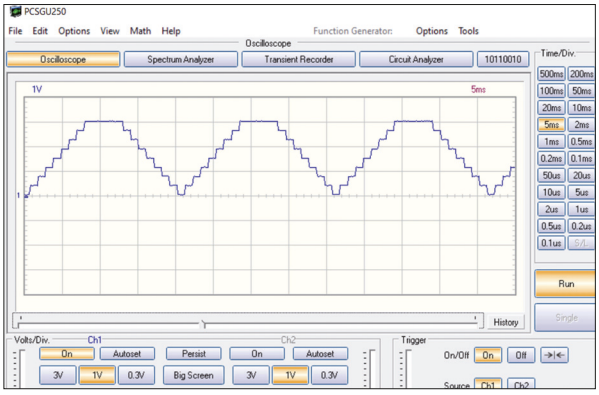

Figure 11.12 Example output waveform

11.7 Project 5 - Generating a sine wave signal

Description: In this project, we will be using the DAC to generate a low frequency sine wave using the built-in trigonometric **sin** function. The generated sine wave will have an amplitude of 1.5 V, a frequency of 100 Hz (period = 10 ms), and an offset of 1.5 V.

The block diagram and circuit diagram of the project are as in Figure 11.2 and Figure 11.3

Program Listing: The frequency of the sine wave to be generated is 100 Hz. This wave has a period of 10 ms, or 10,000 µs. If we assume that the sine wave will consist of 100 samples, then each sample should be output at 10,000/100 = 100 µs intervals. The sample values will be calculated using the trigonometric **sin** function of Python.

The **sin** function will have the format:

$$\sin\left(\frac{2\pi \bullet count}{T}\right)$$

where T is the period of the waveform and is equal to 100 samples. Thus, the sine wave is divided into 100 samples and each sample is output at 100 µs. The above formula can be rewritten as:

$$\sin\left(0.0628 \bullet count\right)$$

It is required that the amplitude of the waveform should be 1.5 V. With a reference voltage of +3.3 V and a 12-bit DAC converter (0 to 4095 quantization levels), 1.5 V is equal to 1.5 × 4095/3.3, which is equal to 1861.3 (i.e. the amplitude). Thus, we will multiply our sine function with the amplitude at each sample to give:

$$1861.3 \bullet \sin\left(0.0628 \bullet count\right)$$

The D/A converter used in this project is unipolar and cannot output negative values. Therefore, an offset is added to the sine wave to shift it so that it is always positive. The offset should be larger than the absolute value of the maximum negative value of the sine wave, which is 1861.3 when the **sin** function above is equal to 1.5. In this project, we are adding a 1.5 V offset which corresponds to a decimal value of 1861.3 (i.e. the offset) at the DAC output. Thus, for each sample, we will calculate and output the following value to the DAC:

$$1861.3 \; + \; 1861.2 \bullet \sin\left(0.0628 \bullet count\right)$$

The sine waveform values for a period are obtained outside the program loop using the following statement. The list **sins** contains all the 100 sine values of the waveform. The reason for calculating these values outside the program loop is to minimize the time to calculate the **sin** function:

```
for i in range(100):
    sins[i] = int(offset + amplitude * sin(R*i)
```

where R is set to 0.0628

Figure 11.13 shows the program listing (program: **sine.py**). Most parts of the program are similar to the other waveform generation programs. Inside the program loop, samples of the sine wave are sent to the DAC at each sample time.

```
#-----------------------------------------------------------------
#                GENERATE SINE WAVEFORM
#                ======================
#
# This program generates sine waveform with a period of 10ms. Both
# the amplitude and the offset of the waveform are set to 1.5V
#
# Author: Dogan Ibrahim
# File  : sine.py
# Date  : October, 2023
#-----------------------------------------------------------------
from gpiozero import LED
from time import sleep                          # Import time
import spidev                                   # Import SPI
import math                                     # Import math

spi = spidev.SpiDev()
spi.open(0, 0)                                  # Bus=0, device=0
spi.max_speed_hz = 3900000

CS = LED(26)
CS.on()                                         # Disable CS

sample = 0
T = 100
R = 0.0628
amplitude = 1861.3
offset = 1861.3
sins = [None]*101

#
# This function implements the DAC. The data in "data" is sent
# to the DAC
#
def DAC(data):
   CS.off()                                     # Enable CS
#
# Send HIGH byte
#
   temp = (data >> 8) & 0x0F                    # Get upper byte
   temp = temp + 0x30                           # OR with 0x30
   spi.xfer2([temp])                            # Send to DAC
```

```
#
# Send LOW byte
#
   temp = data & 0xFF                          # Get lower byte
   spi.xfer2([temp])                           # Send to DAC

   CS.on()                                     # Disable CS

#
# Generate the 100 sine wave samples and store in list sins
#
for i in range(100):
   sins[i] = int(offset + amplitude*math.sin(R*i))

try:
     while True:
        DACValue = sins[sample]                # Value  to send
        DAC(DACValue)                          # Send to DAC
        sleep(0.0001)                          # Wait
        sample = sample + 1                    # Next sample
        if sample == 100:                      # 100 samples?
            sample  = 0

except KeyboardInterrupt:
    pass
```

Figure 11.13 Program listing

An example output waveform taken from the oscilloscope is shown in Figure 11.14. Notice that the frequency of the waveform is not very accurate because the delay function of Python is not accurate.

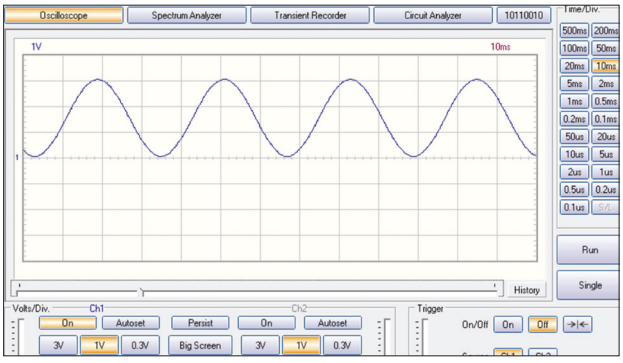

Figure 11.14 Example output waveform

Chapter 12 • Using the Sense HAT

12.1 Overview

The Sense HAT is a small plug-in board developed by Raspberry Pi in collaboration with the UK Space Agency and the European Space Agency (ESA). The board includes a number of sensors and that's why it is called 'Sense'. The word 'HAT' stands for 'Hardware Attached on Top' to indicate that the board is attached or plugged in on top of the Raspberry Pi. Sense HAT gives the flexibility to carry out various environmental measurements using its built-in sensors, and the board was specially developed for the Astro Pi Challenge and competition. An emulator-based version of the Sense HAT is also available to enable students to carry out experiments without having the physical board.

The Sense HAT board has the following features:

- 8 × 8 RGB LED matrix, having 15-bit colour resolution
- Five-button joystick with left, right, up, down, and enter movements
- Gyroscope (angular rate sensor): ±245/500/2000 dps
- Accelerometer (linear acceleration sensor): ±2/4/8/16 G
- Magnetometer (magnetic sensor): ±4/8/12/16 gauss
- Barometer: 260-1260 hPa
- Temperature sensor (with barometer): accuracy ±2ºC in the 0–65ºC range
- Relative humidity sensor: accuracy ±4.5% in the 20–80% range
- Temperature sensor (with humidity): accuracy ±0.5ºC in the 15–40ºC range
- Graphics controller chip

12.2 The Sense HAT interface

The Sense HAT board (Figure 12.1) consists of 7 main components and an LED matrix. The components on the board are controlled via the I²C bus interface. The following are the main components on the board:

Component	I²C bus address	Function
HTS221	0x5F	humidity sensor
LPS254H	0x5C	Pressure/temperature sensor
LSM9DS1	0x1C,0x6A	Accelerometer+magnetometer
SKRHABE010	-	joystick
LED2472G	0x46	LED matrix controller
LED matrix	-	-
ATTINY88	-	Microchip microcontroller

The Sense HAT board is normally plugged into the 40-way connector of the Raspberry Pi. To interface external components to the Raspberry Pi in addition to the Sense HAT board, you need to connect the Sense HAT to the Raspberry Pi using either a ribbon cable or jumper wires so that other pins of the Raspberry Pi can be accessed. Additionally, if you are using an active cooler on your Raspberry Pi 5 then it is not possible to connect the Sense HAT to your board unless you use a ribbon cable or 2×20 pin header extension. Therefore, it is useful to know which pins of the Sense HAT board are used by Raspberry Pi 5, and which

pins are free so that you can make connections to the Sense HAT using jumper wires.

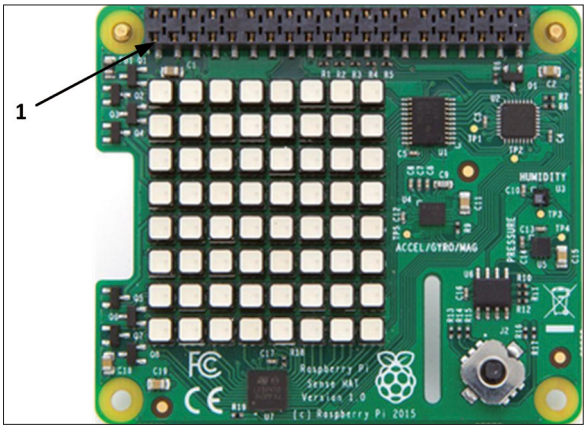

Figure 12.1 The Sense HAT board

In addition to the I²C control lines, the ATTINY88 microcontroller on the board can be programmed via the SPI bus control lines (MOSI, MISO, SCK, CE0) provided on the board.

The following pins are used by the Sense HAT 40-way connector:

Pin number	Raspberry Pi 5 port	Function
3	GPIO2	SDA (I²C)
5	GPIO3	SCL (I²C)
1	+3.3 V	power
19	GPIO10	MOSI (SPI)
21	GPIO9	MISO (SPI)
23	GPIO11	SCK (SPI)
24	GPIO8	CE0 (SPI
9	GND	power ground
2	+5 V	power
16	GPIO23	INT
18	GPIO24	INT
22	GPIO25	PROG
27	ID_SD	EEPROM
28	ID_SC	EEPROM

The Sense HAT board can be connected to your Raspberry Pi 5 using only the following 9 pins of the 40-way connector:

Sense HAT pin	Raspberry Pi 5 Pin	Function
3	3 (GPIO2)	SDA (I²C)
5	5 (GPIO3)	SCL (I²C)
1	1 (+3.3 V)	power
9	9 (GND)	power ground

2	2 (+5 V)	power
16	16 (GPIO23)	joystick
18	18 (GPIO24)	joystick
27	27 (ID_SD)	EEPROM
28	28 (ID_SC)	EEPROM

Note: You can also plug in the Sense HAT board directly on top of your Raspberry Pi 5 board instead of making the above connections provided the GPIO pins are available (e.g. you are not using a Raspberry Pi active cooler)

12.3 Programming the Sense HAT

Sense HAT is installed by default on your latest Raspberry Pi 5 SD card. You may, however, enter the following command to install the latest version of the Sense HAT (at the time of writing this book, the latest version was: 1.4):

```
pi@raspberrypi:~ $ sudo apt-get install sense-hat
```

Before developing a project using the Sense HAT board, the Sense HAT library must be imported into your Python program and also the **sense** object must be created at the beginning of the program. i.e. the following two statements must be included at the beginning of your programs:

```
from sense_hat import SenseHat
sense = SenseHat()
```

The remainder part of this chapter is devoted to developing simple projects with the Sense HAT. In all the projects, the Sense HAT was connected to the Raspberry Pi 5 using jumper wires as described earlier.

12.4 Project 1 – Displaying text on Sense HAT

Description: In this project, you will learn how to display as well as scroll text messages on Sense HAT. The statement **show_message** is used to scroll a text message. In the following code, the message **Sense HAT** is scrolled on the LED matrix. Notice that the message is displayed only once:

```
>>> from sense_hat import SenseHat
>>> sense = SenseHat()
>>> sense.show_message("Sense HAT")
```

If you get an error message saying that the **RPi-Sense FB device cannot be detected**, then do the following:

- pi@raspberrypi:~ $ **sudo nano /boot/config.txt**
- go to the end of the file and enter the following statement:
  dtoverlay=rpi-sense

- Press **CNTRL+X** followed by **Y** to save the change
- Reboot your Raspberry Pi
- pi@raspberrypi:~ $ **sudo reboot now**

Notice that there are two versions of the Sense HAT board. Version 1.0 has no colour sensor, while Version 2.0 has a colour sensor. You may get a warning message saying that it failed to initialize the colour sensor if you are using Version 1.0.

You can also display a single letter using the statement: **sense.show_letter**, for example, **sense.show_letter("A").** Notice that the letter is displayed permanently.

In addition to displaying text in default mode, you can use the following options:

scroll_speed: This floating-point number changes the speed that the text scrolls. The default value is 0.1. A higher number slows down the scroll speed.

text_colour: Used to change the text colour. The colour is specified as (Red, Green, Blue) where each colour can take a value between 0 and 255, and we can mix the colours to obtain any other colour. For example, (255, 0, 0) is red and so on.

back_colour: used to change the colour of the background. Colour is defined as in the text_colour option.

In the following example, the same text as above is scrolled slowly, in red colour, with yellow background colour:

```
>>> from sense_hat import SenseHat
>>> sense = SenseHat()
>>> sense.show_message("Sense HAT", scroll_speed=0.3,

text_colour=[255,0,0], back_colour=[255,255,0])
```

Notice that in the above program, the text is displayed only once, but the background colour remains as yellow.

If, for example, you wish to repeat displaying the text, say every two seconds, then the required program is as shown in Figure 12.2 (program: **txt.py**). Notice how the continuation line is used in Python. Run the program from the Console mode as:

pi@raspberrypi:~ $ **python txt.py**

```
#-------------------------------------------------------------
#                         Display Text
#                          ------------
#
# This program displays the text Sense HAT every 2 seconds.
```

```
# the text colour is RED and back ground colour is YELLOW
#
# Author: Dogan Ibrahim
# File  : txt.py
# Date  : October, 2023
#-------------------------------------------------------------
from sense_hat import SenseHat
import time
sense = SenseHat()

while True:
   sense.show_message("Sense HAT",scroll_speed=0.3,\
text_colour=[255,0,0],back_colour=[255,255,0])
   time.sleep(2)
```

Figure 12.2 Program listing

The **sense.clear()** statement can be used to turn OFF all the LEDs. This may be necessary to ensure that all the LEDs are turned OFF at the beginning of a program. Similarly, a colour can be passed to the clear statement to set all the LEDs to the same colour, such as:

```
red = (255, 0, 0)
sense.clear(red)
```

The brightness of the LED matrix can be changed by toggling the **low_light** statement. In the following examples, the brightness is toggled:

```
sense.low_light = True
```

or

```
sense.low_light = False
```

The displayed text (or image) can be rotated by using the statement **set_rotation(n)** where **n** is the rotation angle in degrees, and it can take the values of 0, 90, 180, 270. The following statement rotates character s by 90 degrees and displays it on the LED matrix:

```
sense.set_rotation(90)
sense.show_letter("s")
```

Text (or image) can be flipped horizontally or vertically by using the statements **flip_h** or **flip_v** respectively. In the following example, the character **X** is flipped horizontally and is then displayed:

```
sense.flip_h
sense.show_letter("X")
```

12.5 Project 2 – Test your math skills - multiplication

Description: This project is aimed for younger readers who may want to test their multiplication skills. The program displays two numbers which are required to be multiplied together. The result of the multiplication is hidden for 10 seconds, and time is given to the user to find the correct answer. After 10 seconds, the correct answer is displayed so that the user can check it against his/her answer. Only numbers from 1 to 99 are considered for simplicity.

Figure 12.3 shows the program listing (program: **mult.py**). Two integer random numbers are generated between 1 and 99 and are stored in variables **no1** and **no2**. Variable **question** holds the question as a string and this is displayed in green colour as shown in the following example:

25 × 10 =

The program waits for 10 seconds and after this time, the result 250 is displayed in red colour. After 2 seconds, the LEDs are cleared, and the program continues displaying two new numbers.

```
#-------------------------------------------------------------
#                      Multiplication Test
#                       -------------------
#
# This program displays two numbers between 1 and 99 and waits
# for 10 seconds until the user finds the correct answer. The
# correct answer is then displayed so that the user can check with
# his/her answer
#
# Author: Dogan Ibrahim
# File  : mult.py
# Date  : October, 2023
#-------------------------------------------------------------
from sense_hat import SenseHat
sense = SenseHat()
import time
import random
spd = 0.2                                    # Scroll speed
red = (255, 0, 0)                            # Red colour
green = (0, 255, 0)                          # Green colour

try:

  while True:
    no1 = random.randint(1,99)               # First number
    no2 = random.randint(1, 99)              # Second number
    question = str(no1) + "x" + str(no2) + "="
```

```
    sense.show_message(question, scroll_speed = spd, text_colour=(green))
    time.sleep(10)
    result = str(no1 * no2)
    sense.show_message(result, scroll_speed = spd, text_colour=(red))
    time.sleep(2)
    sense.clear()
    time.sleep(1)

except KeyboardInterrupt:
    exit()
```

Figure 12.3 Program listing

12.6 Project 3 – Learning the times tables

Description: This project helps the children to practise their times tables. A number (which can be changed) is hard-coded into the program. The program displays the times table for the selected number. For example, if the hard-coded number is 5 then the following is displayed on the LED matrix:

5x1=5
5x2=10
5x3=15
5x4=20
5x5=25
5x6=30
5x7=35
5x8=40
5x9=45
5x10=50
5x11=55
5x12=60

Figure 12.4 shows the program listing (program: **timestab.py**). Variable **Tablefor** stores the number whose times table is required. A loop is formed which goes from 0 to 11. Inside this loop, variable **j** takes on values from 1 to 12. Variable **result** stores the result of the multiplication at each iteration of the loop. String variable **disp** stores the data to be displayed by the LED matrix at each iteration. Users can easily change the value of **Tablefor** to generate times table for another number.

```
#-----------------------------------------------------------------
#                      Times Table
#                       -----------
#
# This program generates a times table. The table is selected at the
# beginning of the program by setting variable Tablefor.
#
```

```
# Author: Dogan Ibrahim
# File   : timestab.py
# Date   : October, 2023
#-------------------------------------------------------------------
from sense_hat import SenseHat
sense = SenseHat()
import time

spd = 0.2                                                # Scroll speed
red = (255, 0, 0)                                        # Red colour
Tablefor = 5                                             # Table for 5

try:

  for k in range(12):                                    # Do 0 to 11
    j = k + 1                                            # 1 to 12
    result = Tablefor * j
    disp = str(Tablefor) + "x" + str(j) + "=" + str(result)
    sense.show_message(disp, scroll_speed = spd, text_colour=(red))
    time.sleep(1)
    sense.clear()

except KeyboardInterrupt:
    exit()
```

Figure 12.4 Program listing

12.7 Project 4 – Display the temperature, humidity, and pressure

Description: In this project we display the ambient temperature, humidity, and pressure on the Sense HAT.

Figure 12.5 shows the program listing (program: **thp.py**). The program runs in a loop every two seconds where the temperature, humidity, and pressure readings are displayed on the scrolling LED. Notice that the readings are all in floating-point format, and the **round()** function is used to configure them to have one digit after the decimal point.

```
#-------------------------------------------------------------
#              TEMPERATURE,HUMIDITY & PRESSURE
#              -------------------------------
#
# This program reads the temperature, humidity and pressure and
# displays on the scrolling LEDs. The data is displayed in the
# following format:
#
#  T=nn.nC H=nn.n% P=nnnn.nmb
#
```

```
# Author: Dogan Ibrahim
# Date  : October, 2023
# File  : thp.py
#-----------------------------------------------------------------
from sense_hat import SenseHat
sense=SenseHat()
import time

while True:
   T = round(sense.get_temperature(), 1)       # Get temperature
   H = round(sense.get_humidity(), 1)          # Get humidity
   P = round(sense.get_pressure(), 1)          # Get pressure
   enviro = "T="+str(T)+ "C H="+str(H)+ "% P="+str(P)+"mb "
   sense.show_message(enviro, scroll_speed = 0.2)
   time.sleep(2)
```

Figure 12.5 Program listing

You could also have displayed the data on the PC screen by running the following program code. Figure 12.6 shows the output of the program:

```
from sense_hat import SenseHat
import time
sense = SenseHat()
while True:
    T = sense.get_temperature()
    H = sense.get_humidity()
    P = sense.get_pressure()
    TT = round(T, 1)
    HH = round(H, 1)
    PP = round(P, 1)
    print("Temperature: %s, Humidity: %s, Pressure:%s"
 %(TT, HH, PP))
    time.sleep(1)
```

```
Temperature: 24.9, Humidity: 43.3, Pressure: 997.8
Temperature: 24.9, Humidity: 43.4, Pressure: 997.8
Temperature: 25.0, Humidity: 43.2, Pressure: 997.8
Temperature: 25.0, Humidity: 43.4, Pressure: 997.8
Temperature: 25.0, Humidity: 43.4, Pressure: 997.8
Temperature: 25.1, Humidity: 43.3, Pressure: 997.9
Temperature: 25.0, Humidity: 43.6, Pressure: 997.8
Temperature: 25.0, Humidity: 43.3, Pressure: 997.9
```

Figure 12.6 Displaying the data on the PC screen

You could also display the temperature or the humidity as integer variables on non-scrolling LEDs by using the **Disp** function.

12.8 Project 5 – ON-OFF temperature controller

Description: This is an on-off temperature controller project. The Sense HAT is connected to the Raspberry Pi 5 to measure the ambient temperature. Additionally, a small buzzer is connected to one of the ports of the Raspberry Pi. The set temperature value is hard-coded in the program. If the ambient temperature is lower than the set temperature, then the buzzer is activated, and the LED matrix displays the ambient temperature in red colour. If, on the other hand, the ambient temperature is higher than the set temperature value, then the buzzer is deactivated, and the ambient temperature is displayed in blue colour. The buzzer in this project can easily be replaced with a relay which can be connected to control a heater. The heater will turn ON if the ambient temperature is lower than the set value.

Block diagram: Figure 12.7 shows the block diagram of the project.

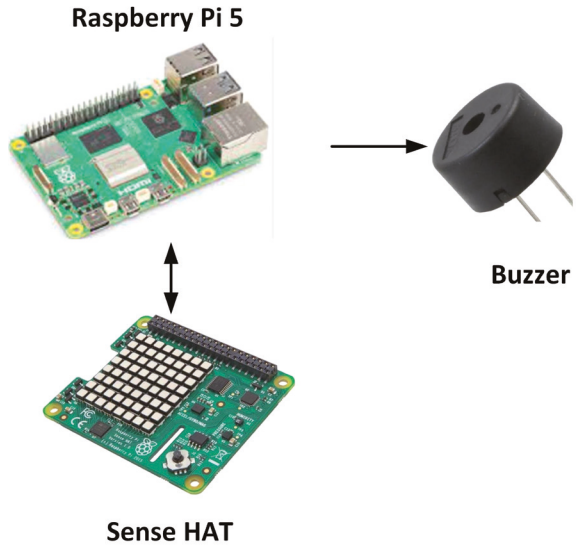

Figure 12.7 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 12.8, where the buzzer is connected to port pin GPIO 4 of the Raspberry Pi 5. Both the buzzer and the Sense HAT board are connected to the Raspberry Pi 5 using jumper wires.

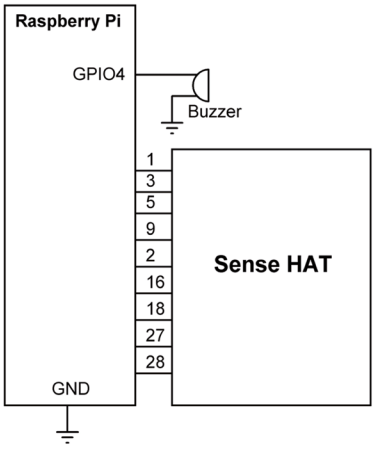

Figure 12.8 Circuit diagram of the project

Program listing: In this program, the library function created by the author, named **Disp()** is used. This function has 3 arguments: number to be displayed, colour, and mode (0 or 1, 1 to clear the display). This function is in a Python program called **display.py**, which can be found on the web page of the book. Calling function **Disp()** display a number without scrolling the display. Figure 12.9 shows the listing of program **display.py**. Make sure that **display.py** is in the same directory as your main program.

```
#------------------------------------------------------------
#                      FUNCTION TO DISPLAY NUMBERS
#                      ---------------------------
#
# This function displays a two-digit number on the LED matrix
# without scrolling the display. The number to be displayed and
# its colour are entered as the arguments of the function.The
# third parameter controls whether or not to clear the display
# before displaying the number. Setting this parameter to 1
# will clear the display
#
# Author: Dogan Ibrahim
# Date  : October, 2023
# File  : display.py
#------------------------------------------------------------
from sense_hat import SenseHat
sense = SenseHat()

def Disp(no, colour, mode):
#
# Number patterns for all the numbers 0 to 9
#
       numbers = [
```

```
        [[0,1,1,0],             # 0
        [1,0,0,1],
        [1,0,0,1],
        [1,0,0,1],
        [1,0,0,1],
        [1,0,0,1],
        [1,0,0,1],
        [0,1,1,0]],

        [[0,0,1,0],             # 1
        [0,1,1,0],
        [0,0,1,0],
        [0,0,1,0],
        [0,0,1,0],
        [0,0,1,0],
        [0,0,1,0],
        [0,1,1,1]],

        [[0,1,1,0],             # 2
        [1,0,0,1],
        [0,0,0,1],
        [0,0,0,1],
        [0,0,1,0],
        [0,1,0,0],
        [1,0,0,0,],
        [1,1,1,1]],

        [[1,1,1,1],             # 3
        [0,0,1,1],
        [0,0,1,1],
        [1,1,1,1],
        [1,1,1,1],
        [0,0,1,1],
        [0,0,1,1],
        [1,1,1,1]],

        [[0,0,1,0],             # 4
        [0,1,1,0],
        [1,1,1,0],
        [1,0,1,0],
        [1,1,1,1],
        [0,0,1,0],
        [0,0,1,0],
        [0,0,1,0]],

        [[1,1,1,1],             # 5
```

```
    [1,0,0,0],
    [1,0,0,0],
    [1,1,1,1],
    [0,0,0,1],
    [0,0,0,1],
    [0,0,0,1],
    [1,1,1,1]],

    [[1,1,1,1],             # 6
    [1,0,0,0],
    [1,0,0,0],
    [1,1,1,1],
    [1,0,0,1],
    [1,0,0,1],
    [1,0,0,1],
    [1,1,1,1]],

    [[1,1,1,1],             # 7
    [0,0,0,1],
    [0,0,0,1],
    [0,0,0,1],
    [0,0,0,1],
    [0,0,0,1],
    [0,0,0,1],
    [0,0,0,1]],

    [[0,1,1,0],             # 8
    [1,0,0,1],
    [1,0,0,1],
    [1,1,1,1],
    [1,0,0,1],
    [1,0,0,1],
    [1,0,0,1],
    [0,1,1,0]],

    [[1,1,1,1],             # 9
    [1,0,0,1],
    [1,0,0,1],
    [1,1,1,1],
    [0,0,0,1],
    [0,0,0,1],
    [0,0,0,1],
    [1,1,1,1]]
    ]

    blank = [0,0,0]
```

```
blanks=[0,0,0,0]
Disp = []                                       # List to store patterns

for index in range(0, 8):
        if (no >= 10):                          # If >= 10
                intno = int(no / 10)            # MSD digit
                Disp.extend(numbers[intno][index])
        else:
                Disp.extend(blanks)
        remno = int(no % 10)                    # LSD digit
        Disp.extend(numbers[remno][index])

for index in range(64):
        if(Disp[index]):
                Disp[index]=colour              # Colour
        else:
                Disp[index]=blank

if mode == 1:
    sense.clear()                               # Clear LEDs

sense.set_pixels(Disp)                          # Display number
```

Figure 12.9 Program display.py

The program listing is shown in Figure 12.10 (program: **tempcont.py**). At the beginning of the program, the modules used in the program are imported to the program. Buzzer is assigned to number 4, which will correspond to GPIO 4. The set temperature value is stored in the variable **SetTemperature** and is hard-coded as 24 in his example. The buzzer is turned OFF at the beginning of the program. The remainder of the program runs in an endless loop. Inside this loop, the ambient temperature is read from the Sense HAT and this temperature is compared with the set point value. If the ambient temperature is less than the set value, then the buzzer is turned ON and the ambient temperature is displayed in red as non-scrolling. If, on the other hand, the ambient temperature is greater than the set value, then the buzzer is turned OFF and the ambient temperature is displayed in blue colour.

```
#-----------------------------------------------------------------------
#              ON-OFF TEMPERATURE CONTROLLER
#              -----------------------------
#
# This is an ON-OFF temperature control project. In this project
# a buzzer is connected to port pin GPIO 4 of the Raspberry Pi 5
# In addition to the Sense HAT. The Sense HAT is connected using
# jumper wires. The buzzer is turned ON if the ambient temperature
# is below the setpoint temperature. At the same time, the ambient
# temperature is displayed in red colour. If on the other hand the
```

```
# ambient temperature is higher than the setpoint value then the
# buzzer is turned OFF and the display is in blue colour.
#
# The buzzer in this program can be replaced with a relay for
# example to control a heater
#
# Author: Dogan Ibrahim
# Date  : October, 2023
# File  : tempcont.py
#-------------------------------------------------------------------------
from gpiozero import LED
from display import Disp                                # import Disp
from sense_hat import SenseHat                          # import Sense HAT
sense=SenseHat()
from time import sleep                                  # import time

Buzzer = LED(4)                                         # Buzzer at GPIO 4
Buzzer.off()                                            # Buzzer off

SetTemperature = 24                                     # setpoint temp
red = (255, 0 ,0)                                       # red colour
blue = (0, 0, 255)                                      # blue colour

while True:
   T = int(sense.get_temperature_from_humidity())       # get temperature
   if(T < SetTemperature):                              # T < setpoint?
      Disp(T, red, 0)                                   # display in red
      Buzzer.on()                                       # Buzzer ON
   else:
      Disp(T, blue, 0)                                  # display in blue
      Buzzer.off()                                      # Buzzer OFF

   sleep(5)                                             # wait 5 secs
```

Figure 12.10 Program listing

The buzzer used in this project can easily be replaced with a relay and a heater can be connected to the heater. The room temperature will then be controlled by the program.

12.9 Project 6 – Generate two dice numbers

Description: Most dice-based games (e.g. backgammon) are played with two dice, where both dice are thrown at the same time. In this project, two random dice numbers are generated and displayed on the LED matrix. The dice numbers are displayed in red.

Figure 12.11 shows the program listing (program: **dice2.py**). Here, two integer random numbers are generated, converted into strings, and stored in variables no1 and no2. Statement **show_message** is used to scroll the generated numbers, where the speed is set to 0.05 and the text colour is set to red. The LED matrix shows the numbers as in the format: 32 24 66 24 etc. as shown in Figure 12.12.

```
#------------------------------------------------------------
#                      Display Two Dice Numbers
#                      ------------------------
#
# This program displays two dice numbers every 5 seconds. The
# numbers are displayed in red
#
# Author: Dogan Ibrahim
# File  : dice2.py
# Date  : October 2023
#------------------------------------------------------------
from sense_hat import SenseHat
sense = SenseHat()
import time
import random
red = (255,0,0)

try:
  while True:
    no1 = str(random.randint(1,6))
    no2 = str(random.randint(1,6))
    no = no1 + no2
    sense.show_message(no, scroll_speed=0.05, text_colour=(red))
    time.sleep(5)
    sense.clear()
    time.sleep(1)
except KeyboardInterrupt:
    exit()
```

Figure 12.11 Program listing

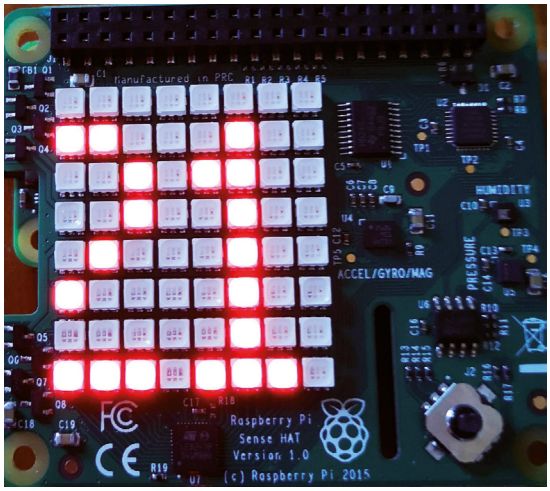

Figure 12.12 Displaying dice numbers 2 and 1

12.10 Project 7 – Display the current time

Description: In this project, the current time is extracted and displayed on the LED matrix in the format: HH:MM:SS. The display is scrolled every second.

Figure 12.13 shows the program listing (program: **curtime.py**). At the beginning of the program, datetime is imported to the program in addition to the other libraries. The scrolling speed is set to 0.15 and the text colour is set to blue. Current time is extracted from the function **datetime.now()** and **strftime** function is used to extract only the hours, minutes, and seconds. The time is updated and displayed every second using **show_message**.

```
#--------------------------------------------------------------------
#                      Display Current Time
#                       ---------------------
#
# This program displays the current time every second on the following
# format:
#         HH:MM:SS
#
# Author: Dogan Ibrahim
# File  : curtime.py
# Date  : October 2023
#--------------------------------------------------------------------
from sense_hat import SenseHat
sense = SenseHat()
import time
import datetime

spd = 0.15                                        # Scroll speed
```

```
blue = (0, 0, 255)                              # Text colour

while True:
    TimeFormat = "%H:%M:%S"
    msg = str(datetime.datetime.now().strftime(TimeFormat))
    sense.show_message(msg,scroll_speed=spd, text_colour=(blue))
    time.sleep(1)
```

Figure 12.13 Program listing

The **strftime()** returns a formatted string representing data and time. Some examples are given below:

```
from datatime import datatime
now = datetime.now()
year = now.strftime("%Y")                       # return current year
month = now.strftime("%m")                      # return current month
date = now.strftime("%Y:%m:%d")                 # return current date
tim = now.strftime("%H:%M:%S")                  # return current time
```

Some other codes that can be used with strftime are (for more details, see link: https://www.programiz.com/python-programming/datetime/strftime):

%A	-	weekday name (e.g. Monday, Tuesday)
%w	-	weekday as a number (e.g. 1, 2)
%d	-	day of the month as zero padded decimal (e.g. 01, 02)
%b	-	month name (e.g. Jan, Feb)
%B	-	full month name (e.g. January, February)
%m	-	month as a zero padded decimal (e.g. 01, 02)
%p	-	AM or M
%H	-	hour as a zero padded decimal, 24-hour clock (e.g. 05, 06)
%I	-	hour as a zero padded decimal, 12-hour clock (e.g. 05, 06)
%y	-	year without century as a zero padded number
%Y	-	year with century

12.11 Project 8 – Displaying two-digit integer numbers

Description: Sense HAT displays two-digit integer numbers by scrolling the display. In some applications, we may want to display a two-digit number without scrolling the display. For example, while displaying the temperature, humidity, etc., you may want a non-scrolling steady display. In this project, a program has been developed that can display a two-digit number without scrolling the display.

Figure 12.14 shows the program listing (program: **dispnum.py**). This program displays the number 20 as an example. At the beginning of the program, the patterns for all the numbers from 0 to 9 are defined. The two digits of the number are extracted and saved in variables **intno** and **remno**. For example, if the number is 20, then **intno** and **remno** are

set to 2 and 0 respectively. The LEDs to be turned ON are then combined in a list called **Disp**. The LED matrix is cleared just before displaying the number. The numbers are displayed in red colour. Figure 12.15 shows the number 20 displayed on the LED matrix.

```
#------------------------------------------------------------
#                      DISPLAY NUMBERS
#                      ---------------
#
# This program displays a two-digit number on the LED matrix
# without scrolling the display. In this example number 20 is
# displayed
#
# Author: Dogan Ibrahim
# Date  : October 2023
# File  : dispnum.py
#------------------------------------------------------------

from sense_hat import SenseHat
sense = SenseHat()

#
# Number patterns for all the numbers 0 to 9
#
numbers = [
[[0,1,1,0],            # 0
[1,0,0,1],
[1,0,0,1],
[1,0,0,1],
[1,0,0,1],
[1,0,0,1],
[1,0,0,1],
[0,1,1,0]],

[[0,0,1,0],            # 1
[0,1,1,0],
[0,0,1,0],
[0,0,1,0],
[0,0,1,0],
[0,0,1,0],
[0,0,1,0],
[0,1,1,1]],

[[0,1,1,0],            # 2
[1,0,0,1],
[0,0,0,1],
[0,0,0,1],
[0,0,1,0],
```

```
[0,1,0,0],
[1,0,0,0,],
[1,1,1,1]],

[[1,1,1,1],           # 3
[0,0,1,1],
[0,0,1,1],
[1,1,1,1],
[1,1,1,1],
[0,0,1,1],
[0,0,1,1],
[1,1,1,1]],

[[0,0,1,0],           # 4
[0,1,1,0],
[1,1,1,0],
[1,0,1,0],
[1,1,1,1],
[0,0,1,0],
[0,0,1,0],
[0,0,1,0]],

[[1,1,1,1],           # 5
[1,0,0,0],
[1,0,0,0],
[1,1,1,1],
[0,0,0,1],
[0,0,0,1],
[0,0,0,1],
[1,1,1,1]],

[[1,1,1,1],           # 6
[1,0,0,0],
[1,0,0,0],
[1,1,1,1],
[1,0,0,1],
[1,0,0,1],
[1,0,0,1],
[1,1,1,1]],

[[1,1,1,1],           # 7
[0,0,0,1],
[0,0,0,1],
[0,0,0,1],
[0,0,0,1],
[0,0,0,1],
[0,0,0,1],
```

```
[0,0,0,1]],

[[0,1,1,0],            # 8
[1,0,0,1],
[1,0,0,1],
[1,1,1,1],
[1,0,0,1],
[1,0,0,1],
[1,0,0,1],
[0,1,1,0]],

[[1,1,1,1],            # 9
[1,0,0,1],
[1,0,0,1],
[1,1,1,1],
[0,0,0,1],
[0,0,0,1],
[0,0,0,1],
[1,1,1,1]]
]

blank = [0,0,0]
blanks=[0,0,0,0]
Disp = []                                   # List to store patterns

no = 20                                     # Number to be displayed

for index in range(0, 8):
    if (no >= 10):                          # If >= 10
        intno = int(no / 10)                # MSD digit
        Disp.extend(numbers[intno][index])
    else:
        Disp.extend(blanks)
    remno = int(no % 10)                    # LSD digit
    Disp.extend(numbers[remno][index])

for index in range(64):
  if(Disp[index]):
     Disp[index]=(255,0,0)                  # Red colour
  else:
    Disp[index]=blank

sense.clear()                               # Clear LEDs
sense.set_pixels(Disp)                      # Display number
```

Figure 12.14 Program listing

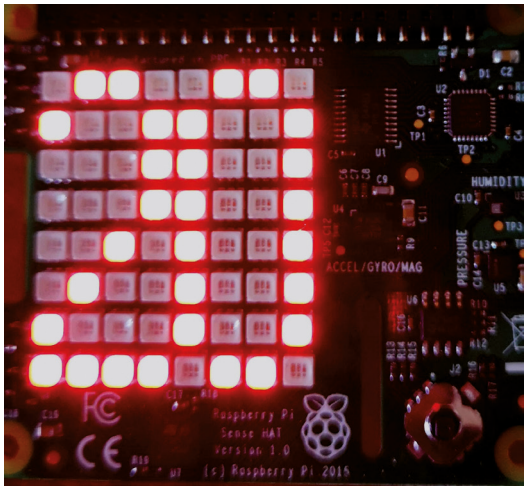

Figure 12.15 Displaying number 20

12.12 Project 9 – Up counter

Description: In this project, the display program developed in the previous project is configured as a function and is then used in a program to count up every second from 0 to 99.

Figure 12.16 shows the program listing (program: **nums.py**). The display function is named **Disp** and is stored in a file called **Display.py** (Figure 12.9). The Function **Disp** has three arguments. The first argument is the number to be displayed, the second argument is the text colour of the display. The third parameter controls whether to clear the display before displaying the number. Setting this parameter to 1 clears the display. At the beginning of the program, the function **Disp** is imported into the program. You should make sure that the Python program **display.py** is in the same directory as the main program **nums.py**. The program creates a loop where variable **j** changes from 0 to 99. Function **Disp** is called with **j** as the number and the colour is set to green. Therefore, the display shows the numbers counting up every second from 0 to 99, without scrolling the display.

```
#--------------------------------------------------------------
#                          UP COUNTER
#                          ----------
#
# This program counts up from 0 to 99 every second and displays
# on the LED matrix without any scrolling
#
# Author: Dogan Ibrahim
# Date  : October, 2023
# File  : nums.py
#--------------------------------------------------------------
from sense_hat import SenseHat
sense = SenseHat()
```

```
from display import Disp                    # Disp function
import time

for j in range(100):                        # Do 0 to 99
  Disp(j, (0,255,0), 1)                     # Display j
  time.sleep(1)                             # 1 sec delay
```

Figure 12.16 Program listing

12.13 The inertial measurement sensor

The Sense HAT contains an Inertial Measurement Unit (IMU) which is a combination of a compass sensor, gyroscope sensor, and accelerometer sensor. These sensors can be enabled or disabled on an individual basis using the **imu_config** statement. For example, in the following example, all three sensors are enabled:

```
sense.set_imu_config(True, True, True)
```

Similarly, if we wish to enable only the gyroscope sensor, we have to use the statement:

```
sense.set_imu_config(False, True, False)
```

12.13.1 Project 10 - Reading the acceleration

Description: You can get the acceleration (amount of G-force) in three dimensions x, y and z by using the statement:

```
x, y, z = sense.get_accelerometer_raw().values()
```

In the example code below, the acceleration in 3 dimensions is read and displayed continuously. You should run this program and move your Sense HAT in three dimensions and see the acceleration changing in each direction:

```
from sense_hat import SenseHat
sense = SenseHat()
while True:
    x, y, z = sense.get_accelerometer_raw().values()
    print("X=%s, Y=%s, Z=%s" %(x, y, z))
```

Rotating the Sense HAT will change the accelerometer x and y values between -1 and +1. If it is placed upside down, the z value will change between -1 and +1. If any axis has ±1 G, then we know that axis is pointing downwards.

An example project is given below.

12.13.2 Project 11 – Accelerometer-based dice

Description: In this example, the shaking of the Sense HAT board is detected, and then a dice number is displayed on the LCD matrix.

If the board is rotated, then the acceleration will be 1 G maximum in any direction. If on the other hand, the board is shaken then the acceleration will be greater than 1 G. In this program, the acceleration in three dimensions is checked to determine when the board is shaken and if so, a dice number is displayed for two seconds.

Figure 12.17 shows the program listing (program: **shake.py**). The dice numbers 1 to 6 are stored in list **dice**. Inside the program loop, the accelerometer values are read, and then their absolute values are taken. If the acceleration exceeds 2 in any direction, i.e. if the board is shaken, a dice number is chosen at random and displayed on the LED matrix in red.

```
#-------------------------------------------------------------
#                    ACCELEROMETER BASED DICE
#                    ------------------------
#
# This program displays a dice number after the Sense HAT board
# is shaken. The accelerometer in 3 dimensions is used to determine
# when the board is shaken
#
# Author: Dogan Ibrahim
# Date  : October, 2023
# File  : shake.py
#-------------------------------------------------------------
from sense_hat import SenseHat
sense=SenseHat()
import time
import random

dice = ['1', '2', '3','4', '5', '6']                  # Dice nos

sense.clear()

while True:
  x, y, z = sense.get_accelerometer_raw().values()     # Read acc
  x = abs(x)                                           # x val
  y = abs(y)                                           # y val
  z = abs(z)                                           # z val
  if x > 2 or y > 2 or z > 2:
    sense.show_letter(random.choice(dice), text_colour = (255,0,0))
    time.sleep(2)
    sense.clear()
```

Figure 12.17 Program listing

12.13.3 Project 12 – Accelerometer-based LED shapes

Description: In this project, the accelerometer is used to sense when the Sense HAT board is tilted in its pitch and roll axes. Initially, the LED at the center of the board is lit. By tilting the board on its axes, you can make turn ON other LEDs and make various interesting shapes.

Figure 12.18 shows the program listing (program: **shapes.py**). The reason for using the accelerometer in this project is because it is more reliable and gives consistent results. If any axis has ±1 G, then we know that axis is pointing downwards. When the Sense HAT board is tilted in its pitch axis, the LEDs in the x-direction are turned ON depending on whether the tilt is in the positive or the negative direction (+G or –G). Similarly, when the board is tilted in its roll axis, the LEDs in the y-direction are turned ON depending on whether the tilt is in the positive or the negative direction.

```
#--------------------------------------------------------------
#                    ACCELEROMETER BASED LED SHAPES
#                    ------------------------------
#
# In this program the accelerometer is used. The Sense HAT board
# is tilted in its pitch and roll axes to make shapes with LEDs
#
# Author: Dogan Ibrahim
# Date  : October, 2023
# File  : shapes.py
#--------------------------------------------------------------
from sense_hat import SenseHat
sense=SenseHat()
from time import sleep

sense.clear()
sense.set_pixel(3, 3, (255,0,0))                        # Starting LED
x = 3
y = 3

while True:
  X,Y,Z = sense.get_accelerometer_raw().values()        # Read acc
  X = round(X, 0)                                       # X dir
  Y = round(Y, 0)                                       # Y dir

  if X > 0:
     x = x + 1
     if x > 7:                                          # If the end
      x = 7
  elif X < 0:
     x = x - 1
```

```
    if x < 0:                                    # If the end
      x = 0

if Y > 0:
    y = y + 1
    if y > 7:                                    # If the end
       y = 7
elif Y < 0:
    y = y - 1
    if y < 0:                                    # If the end
      y = 0
sense.set_pixel(x,y,(255,0,0))                   # Display
sleep(1)                                         # 1 sec delay
```

Figure 12.18 Program listing

Figure 12.19 shows an example shape drawn by tilting the Sense Hat board.

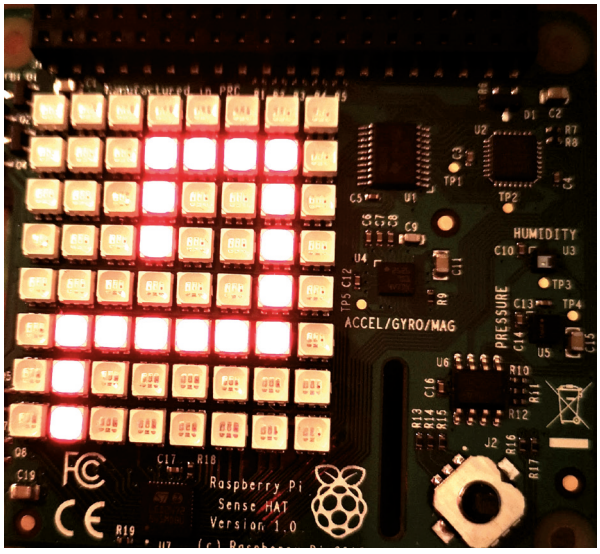

Figure 12.19 Example drawing shapes on the Sense HAT board

Chapter 13 • Using a 4×4 Keypad

13.1 Overview

Keypads are useful devices for entering data to microcontroller-based systems. They are especially useful in portable applications where the user has to enter data or make a choice. In this chapter, you will be learning to use a 4×4 keypad in your Raspberry Pi 5 projects.

13.2 Project 1 – Using a 4×4 keypad

Description: This is a 4×4 keypad program. The program reads the key pressed by the user and displays its code on the screen. The aim of the project is to show how a 4×4 keypad can be used with a Raspberry Pi 5 project.

The 4×4 Keypad: There are several types of keypads that can be used in microcontroller-based projects. In this project, a 4×4 keypad (see Figure 13.1) is used. This keypad has keys for numbers 0 to 9 and the letters A, B, C, D, *, and #. The keypad is interfaced to the processor with 8 wires with the names R1 to R4 and C1 to C4, representing the rows and columns respectively of the keypad (see Figure 13.2).

Figure 13.1 4×4 keypad

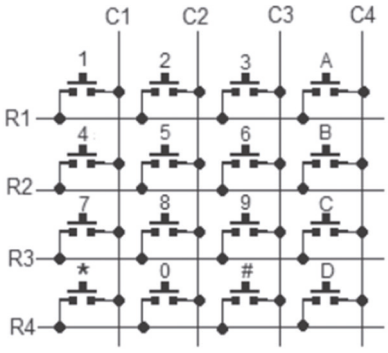

Figure 13.2 Circuit diagram of the 4×4 keypad

The operation of the keypad is basic: the columns are configured as inputs, and they are all set HIGH, and the rows are configured as outputs. The pressed key is identified by using column scanning. Here, a row is forced LOW while the other rows are held HIGH. Then the state of each column is scanned, and if a column is found to be LOW, then the intersection of that column and row is the key pressed. This process is repeated for all the rows.

Block diagram: Figure 13.3 shows the block diagram

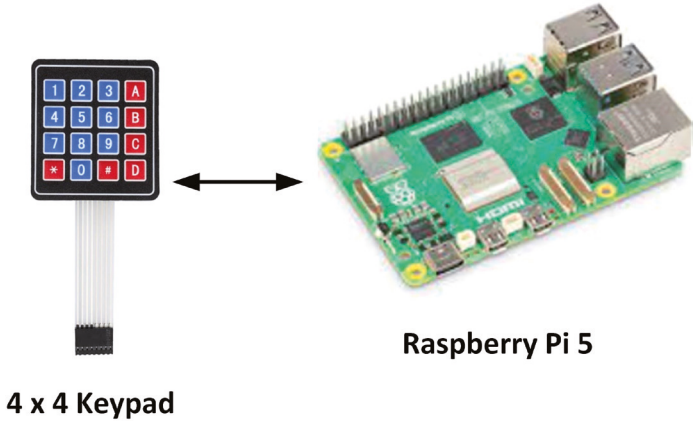

Figure 13.3 Block diagram

Circuit diagram: The circuit diagram of the project is shown in Figure 13.4. The 4×4 keypad is connected to the following GPIO pins of the Raspberry Pi 5. The column pins are held high by using external 10 Kilo-ohm resistors to +3.3 V:

Keypad pin	Raspberry Pi pin
R1	GPIO 4
R2	GPIO 17
R3	GPIO 27
R4	GPIO 22
C1	GPIO 10
C2	GPIO 9
C3	GPIO 11
C4	GPIO 0

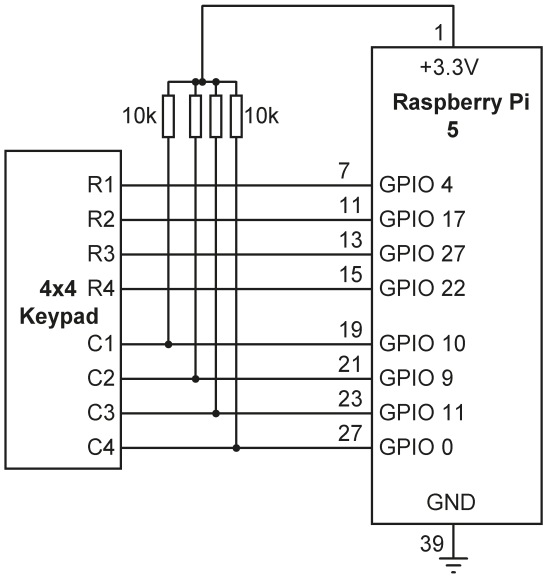

Figure 13.4 Circuit diagram

Figure 13.5 shows the pin configuration of the 4×4 keypad used in the project.

Figure 13.5 Pin configuration of the 4×4 keypad

Program listing: Figure 13.6 shows the program listing (program: **keypad.py**). At the beginning of the program, modules **OutputDevice** and **InputDevice** of **gpiozero** are imported to the program. The row and column pins of the keypad are assigned to GPIO ports. Rows are configured as outputs and columns as inputs. All the rows are set HIGH initially. The Function **GetChar()** waits until a key is pressed and then returns the key to the calling code. This function calls function **ReadRow()**. **ReadRow()** has two arguments: the row

number and the keypad characters on that row. The function scans the columns and if a column is detected with LOW state, then the keypad character corresponding to the column is returned by the function. The program calls **GetChar()** and displays the pressed key on the screen.

```
#-----------------------------------------------------------------
#
#               4 x 4 KEYPAD
#               ============
#
# In this program a 4 x 4 keypad is connected to Raspberry Pi 5.
# the program displays the key pressed on the screen
#
# Program: keypad.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#-----------------------------------------------------------------
from gpiozero import OutputDevice,InputDevice
from time import sleep

#
# ROW pins
#
ROW1 = 4
ROW2 = 17
ROW3 = 27
ROW4 = 22

#
# COLUMN pins
#
COL1 = 10
COL2 = 9
COL3 = 11
COL4 = 0

#
# ROWS as outputs
#
row1 = OutputDevice(ROW1)
row2 = OutputDevice(ROW2)
row3 = OutputDevice(ROW3)
row4 = OutputDevice(ROW4)
row1.on()
row2.on()
row3.on()
```

```
row4.on()

#
# COLUMNS as inputs and (pulled HIGH in hardware)
#
col1 = InputDevice(COL1)
col2 = InputDevice(COL2)
col3 = InputDevice(COL3)
col4 = InputDevice(COL4)

#
# This function sets a row  to 0 and then finds out which
# key is pressed on a column
#
def ReadRow(line, char):
   x = 'E'
   line.off()
   if col1.value == 0:
      x = char[0]
   if col2.value == 0:
      x = char[1]
   if col3.value == 0:
      x = char[2]
   if col4.value == 0:
      x = char[3]
   line.on()
   return x

#
# This function waits until a character is pressed on keypad
#
def GetChar():
  r = 'E'
  while r == 'E':
     a = ReadRow(row1, ["1","2","3","A"])
     b = ReadRow(row2, ["4","5","6","B"])
     c = ReadRow(row3, ["7","8","9","C"])
     d = ReadRow(row4, ["*","0","#","D"])
     if  a != 'E':
        r = a
     elif b !='E':
        r = b
     elif c != 'E':
        r = c
     elif d != 'E':
        r = d
```

```
    sleep(0.1)
  return r

c = GetChar()                  # Wait for key press
print (c)                      # Display the pressed key
```

Figure 13.6 Program listing

Importing the keypad functions in a program

It is easier to import the keypad function under a file instead of writing them every time you want to use these functions. This can easily be done by collecting all the functions in a file and then importing that file at the beginning of your Python programs. Figure 13.7 shows a program called **keypadfuncs.py** which can be imported into your programs. It is important that this file should be in your default directory (**/home/pi**). Note that the keypad rows and columns must be connected to the same Raspberry Pi 5 GPIO pins as given in this project.

```
#-----------------------------------------------------------------
#               4 x 4 KEYPAD FUNCTIONS
#               ======================
#
# Import this file in your Python programs
#
# Program: keypadfuncs.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#-----------------------------------------------------------------
from gpiozero import OutputDevice,DigitalInputDevice
from time import sleep

#
# ROW pins
#
ROW1 = 4
ROW2 = 17
ROW3 = 27
ROW4 = 22

#
# COLUMN pins
#
COL1 = 10
COL2 = 9
COL3 = 11
COL4 = 0
```

```
#
# ROWS as outputs
#
row1 = OutputDevice(ROW1)
row2 = OutputDevice(ROW2)
row3 = OutputDevice(ROW3)
row4 = OutputDevice(ROW4)
row1.on()
row2.on()
row3.on()
row4.on()

#
# COLUMNS as inputs and (pulled HIGH in hardware)
#
col1 = DigitalInputDevice(COL1, bounce_time = 1)
col2 = DigitalInputDevice(COL2, bounce_time = 1)
col3 = DigitalInputDevice(COL3, bounce_time = 1)
col4 = DigitalInputDevice(COL4, bounce_time = 1)

#
# This function sets a row  to 0 and then finds out which
# key is pressed on a column
#
def ReadRow(line, char):
   x = 'E'
   line.off()
   if col1.value == 0:
      x = char[0]
   if col2.value == 0:
      x = char[1]
   if col3.value == 0:
      x = char[2]
   if col4.value == 0:
      x = char[3]
   line.on()
   return x

#
# This function waits until a character is pressed on keypad
#
def GetChar():
  r = 'E'
  while r == 'E':
     a = ReadRow(row1, ["1","2","3","A"])
     b = ReadRow(row2, ["4","5","6","B"])
```

```
    c = ReadRow(row3, ["7","8","9","C"])
    d = ReadRow(row4, ["*","0","#","D"])
    if  a != 'E':
       r = a
    elif b !='E':
       r = b
    elif c != 'E':
       r = c
    elif d != 'E':
       r = d
    sleep(0.1)
  return r
```

Figure 13.7 Program: keypadfuncs.py

Figure 13.8 shows a program (**keypadtest.py**) that imports the keypad functions.

```
#-----------------------------------------------------------------
#
#               4 x 4 KEYPAD TEST
#               =================
#
# This program imports the keypad functions
#
# Program: keypadtest.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#-----------------------------------------------------------------
from keypadfuncs import GetChar

c = GetChar()                  # Wait for key press
print (c)                      # Display the pressed key
```

Figure 13.8 Program: keypadtest.py

13.3 Project 2 – Security lock with keypad and LCD

Description: This is an electronic lock project where a relay is used to open a door. A 4-digit secret code is set up in the program. The user has to enter the secret code for the door to open.

Block diagram: Figure 13.9 shows the block diagram of the project.

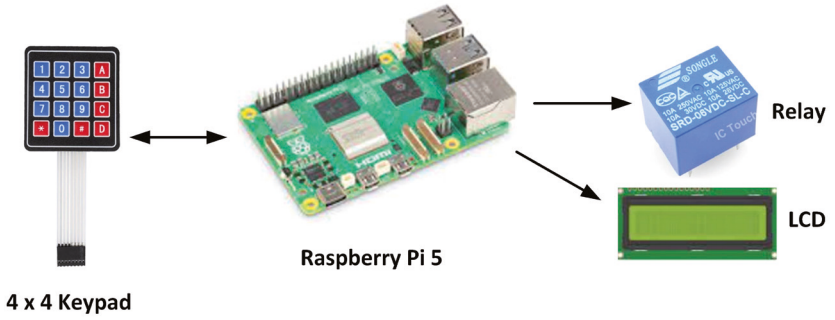

Figure 13.9 Block diagram

Circuit diagram: The circuit diagram is shown in Figure 13.10. The LCD is connected as in the previous LCD-based projects. The keypad is connected as in the previous project. A relay is connected to GPIO 21 (pin 40) of the Raspberry Pi 5.

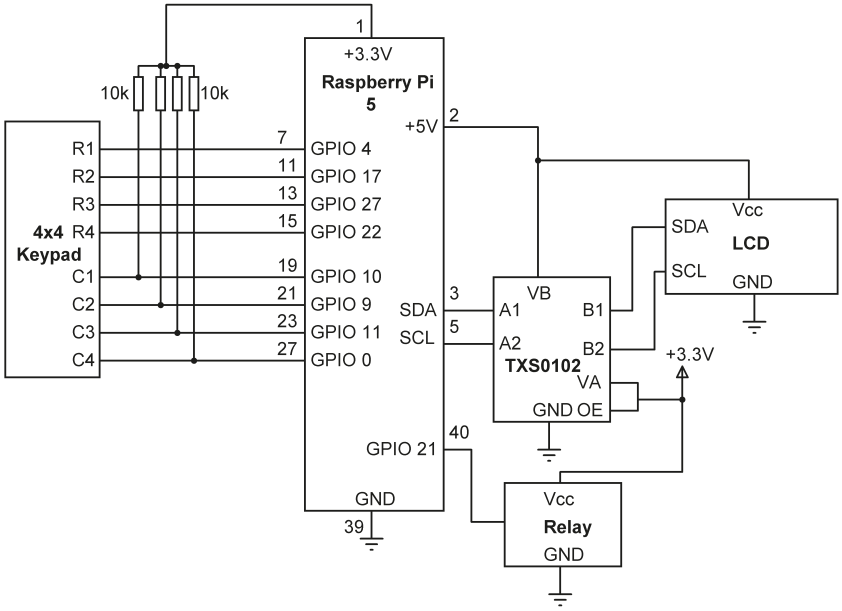

Figure 13.10 Circuit diagram

Program listing: Figure 13.11 shows the program listing **(lock.py**). At the beginning of the program, the LCD is initialized. The secret code is set to '1357'. The program then displays **Code:** and expects the user to enter the correct code. If the correct code is entered, the message **Door Opened** is displayed and the relay is turned ON for 20 seconds. After this time, the relay is deactivated. If the wrong code is entered, the message **Error** is displayed for 5 seconds and the user is asked to enter the correct code again.

```
#-------------------------------------------------------
#
#             KEYPAD OPERATED LOCK
#             ====================
# In this program a door (or a safe) is opened via a relay.
# The user is required to enter the correct secret code for
# the door to open. Once opened, the door stays open for
# 20 seconds
#
# Program: lock.py
# Date   : October, 2023
# Author : Dogan Ibrahim
#-------------------------------------------------------
from time import sleep
from lcd_api import LcdApi
from i2c_lcd import I2cLcd
from keypadfuncs import GetChar
from gpiozero import OutputDevice

Relay = OutputDevice(21)
Relay.off()

I2C_ADDR = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

mylcd = I2cLcd(1,I2C_ADDR,I2C_NUM_ROWS,I2C_NUM_COLS)

mylcd.clear()                           # clear LCD

Codea = "1"                             # Secret code
Codeb = "3"
Codec = "5"
Coded = "7"

while True:
  mylcd.move_to(0,0)
  mylcd.putstr("Code: ")
  a = GetChar()                                 # First no
  b = GetChar()                                 # Second no
  c = GetChar()                                 # Third no
  d = GetChar()                                 # Fourth no

  if (a == Codea and b == Codeb and c == Codec and d == Coded):
     mylcd.clear()
     mylcd.putstr("Door Opened")
```

```
        Relay.on()
        sleep(20)
        Relay.off()
        mylcd.clear()
    else:
        Relay.off()
        mylcd.clear()
        mylcd.putstr("Error")
        sleep(5)
        mylcd.clear()
```

Figure 13.11 Program listing

Suggested modification: Modify the program in Figure 13.11 so that the lock is disabled for ten minutes if the wrong code is entered three times.

Chapter 14 • Communication over Wi-Fi

14.1 Overview

Perhaps the two major features of the Raspberry Pi 5 are its Wi-Fi and Bluetooth communication capabilities. Raspberry Pi 5 is equipped with a dual-band 2.4 GHz 802.11ac wireless LAN module and Bluetooth 5.0/Bluetooth Low Energy (BLE). Without such features, you have to use external network-based hardware communication modules to communicate over the Internet. Network communication is handled using either UDP or TCP type protocols. In this chapter, you will be learning how to write Python programs using both the UDP and TCP type protocols using the on-board Wi-Fi module.

14.2 UDP and TCP

Communication over a Wi-Fi link is in the form of client and server, and sockets are used to send and receive data packets. The server side usually waits for a connection from the clients and once a connection is made, two-way communication can start. Two protocols are mainly used for sending and receiving data packets over a Wi-Fi link: UDP and TCP. TCP is a connection-based protocol which guarantees the delivery of packets. Packets are given sequence numbers and the reception of all the packets is acknowledged to avoid them arriving in the wrong order. As a result of this confirmation, TCP is usually slow, but it is reliable as it guarantees the delivery of packets. UDP, on the other hand, is not connection-based. Packets do not have sequence numbers and as a result of this, there is no guarantee that the packets will arrive at their destinations, or they may arrive in the wrong order. UDP has less overhead than TCP and, as a result, it is faster. Table 14.1 lists some of the differences between the TCP and UDP protocols.

TCP	UDP
Packets have sequence numbers and delivery of every packet is acknowledged	There is no delivery acknowledgement
Slow	Fast
No packet loss	Packets may be lost
Large overhead	Small overhead
Requires more resources	Requires fewer resources
Connection based	Not connection based
Not suitable for multicast	Has multicast capability
More difficult to program	Easier to program
Examples: HTTP, HTTPS, FTP	Examples: DNS, DHCP, Computer games

Table 14.1 TCP and UDP packet communications

14.2.1 UDP communication

Figure 14.1 shows the UDP communication over a Wi-Fi link:

Server

1. Create UDP socket
2. Bind the socket to server address
3. Wait until datagram packet arrives from the client
4. Process the datagram packet
5. Send a reply to the client, or close the socket
6. Go back to Step 3 (if not closed)

Client

1. Create UDP socket (and optionally Bind)
2. Send a message to the server
3. Wait until a response from the server is received
4. Process reply
5. Go back to step 2, or close the socket

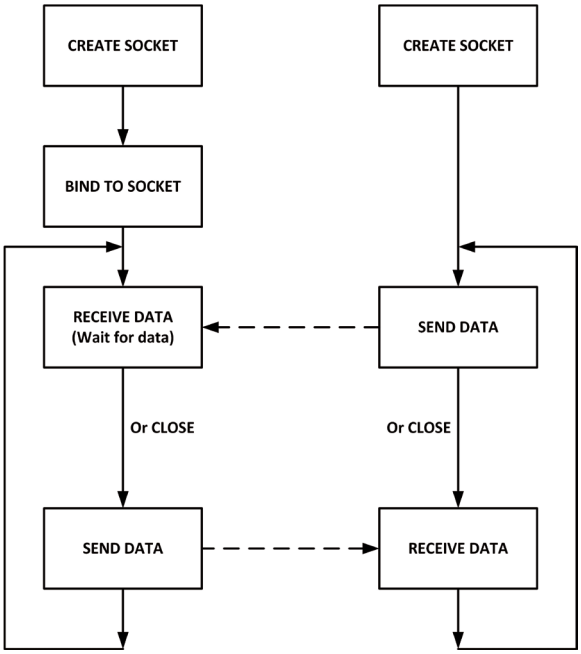

Figure 14.1 UDP communication

14.2.2 TCP communication

Figure 14.2 shows the TCP communication over a Wi-Fi link:

Server

1. Create UDP socket

2. Bind the socket to server address
3. Listen for connections
4. Accept connection
5. Wait until datagram packet arrives from the client
6. Process the datagram packet
7. Send a reply to the client, or close the socket
8. Go back to Step 3 (if not closed)

Client

1. Create UDP socket
2. Connect to the server
3. Send a message to the server
4. Wait until a response from the server is received
5. Process reply
6. Go back to step 2, or close the socket

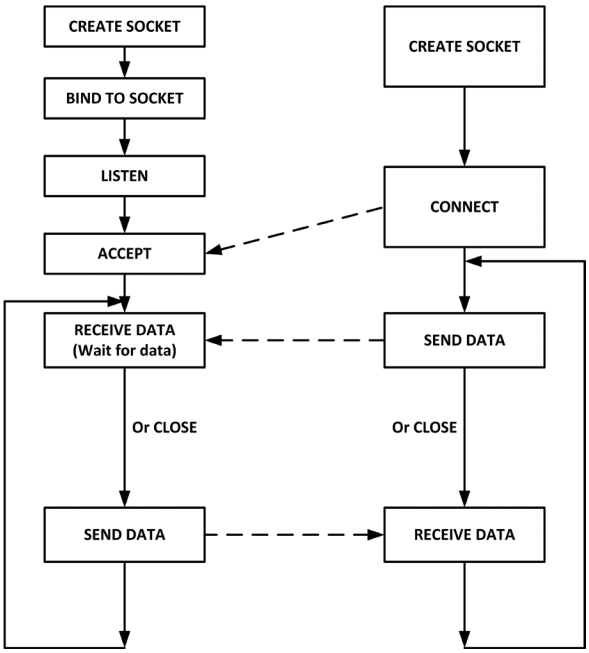

Figure 14.2 TCP communication

14.3 Project 1 – Sending a text message to a smartphone using TCP/IP

Description: In this project, a TCP/IP-based communication is established with an Android smartphone. The program reads text messages from the keyboard and sends to the smartphone. The aim of this project is to show how TCP/IP communication can be established with an Android smartphone.

Background Information: Port numbers range from 0 to 65,535. Numbers from 0 to 1023 are reserved and are called as well-known ports. For example, port 23 is the Telnet port, port 25 is the SMTP port, etc. In this section, you will be using port number 5000 in your program.

Block diagram: Figure 14.3 shows the project block diagram where the Raspberry Pi 5 and smartphone communicate over a Wi-Fi router.

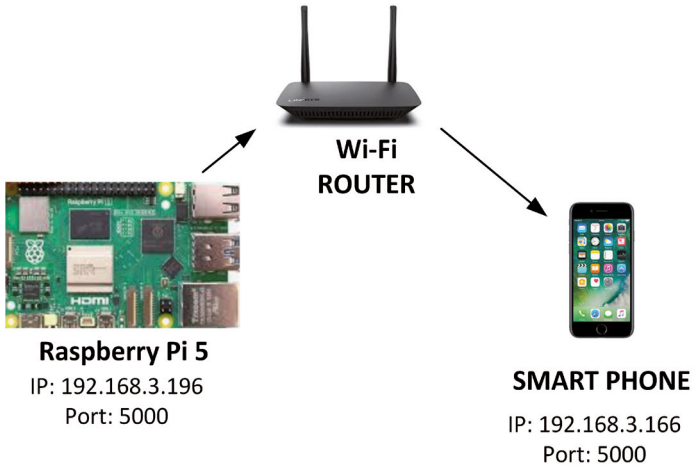

Figure 14.3 Block diagram of the project

Program listing: In this project, Raspberry Pi 5 is the server. Figure 14.4 shows the program listing (**tcpserver.py**). At the beginning of the program, a TCP/IP socket is created (**sock.SOCK_STREAM**) and is then bind to port 5000. The program listens for a connection. Notice that it is possible for the server to listen to multiple clients, but of course, it can communicate with only one at any time. When the client makes a connection, this is accepted by the server. The server then reads a message from the keyboard and sends it to the client over the Wi-Fi link. Notice that the **setsockopt()** statement makes sure that the program can be used again without having to wait for the socket timeout of 30 seconds.

```
#================================================================
#      SEND TEXT MESSAGES USING TCP/IP
#      ===============================
#
# This is the TCP/IP server program. It receives text messages
# from the keyboard and sends to an Android smart phone over
# a Wi-Fi link
#
# Author: Dogan Ibrahim
# File  : tcpserver.py
# Date  : October, 2023
#================================================================
```

```
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(("192.168.3.196", 5000))
sock.listen(1)

client, addr = sock.accept()                    # accept connection
print("Connected to client: ", addr)            # connected message

yn = 'y'

while yn == 'y':
  msg = input("Enter your message: ")           # read a message
  client.send(msg.encode('utf-8'))              # send the message

  yn = input("Send more messages?: ")
  yn = yn.lower()

print("\nClosing connection to client")
sock.close()
```

Figure 14.4 Program listing

Testing

There are many TCP apps available free of charge on the Internet for smartphones. In this project, the **TCP Client by JOY S.R.L.** apps is used on an Android smartphone. This app is available free of charge in the **Play Store** (see Figure 14.5).

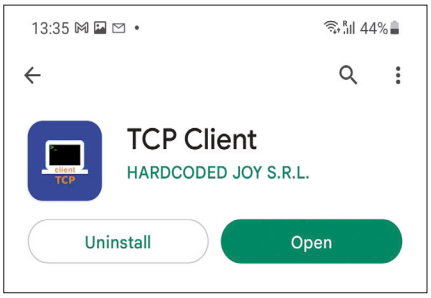

Figure 14.5 Apps used in the project

The program is run as follows:

- Run the server program first:

  pi@raspberrypi:~ $ **python tcpserver.py**

- Run the Android apps and configure it as shown in Figure 14.6 (click the settings icon at the top right hand of the screen), where 192.168.3.196 is the IP address of the Raspberry Pi 5.

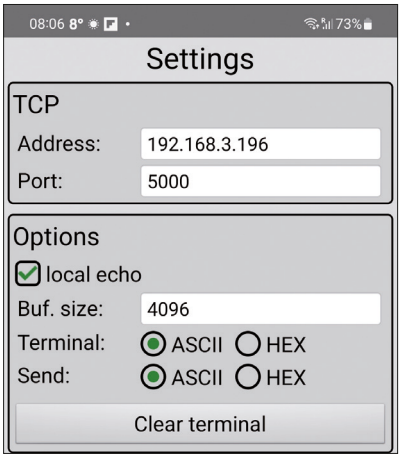

Figure 14.6 Configure the TCP Client apps

- Click the icon in the top left corner of the apps (disconnected) to connect to Raspberry Pi 5 over TCP/IP.

- You should see a connection message on your Raspberry Pi screen and also the IP address of the remote Android smartphone. Now enter a message and press the Enter key. In this example, the message **HELLO FROM RASPBERRY PI** is sent to the client (Figure 14.7). Figure 14.8 shows the message displayed on the smartphone.

```
pi@raspberrypi:~ $ python tcpserver.py
Connected to client:  ('192.168.3.166', 35406)
Enter your message: HELLO FROM RASPBERRY PI
Send more messages?: n

Closing connection to client
pi@raspberrypi:~ $
```

Figure 14.7 Enter the message on the keyboard

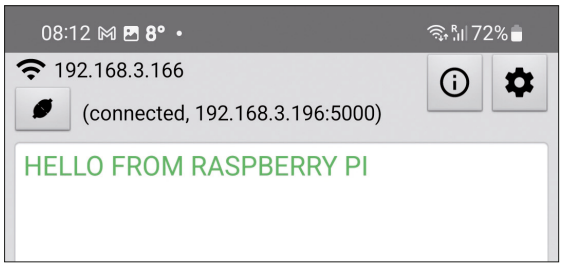

Figure 14.8 Message displayed on smartphone

14.4 Project 2 – Two-way communication with the smartphone using TCP/IP

Description: This project is similar to the previous one, but here two-way communication is established between the Raspberry Pi 5 and the smartphone.

The block diagram of the project is the same as Figure 14.3

Program listing: Figure 14.9 shows the program listing (**tcp2way.py**). Here, port 5000 is used as in the previous project. The program has been changed to send and receive messages from the smartphone. Socket function **recv(byte count)** sends a message over the TCP/IP link to the connected node.

```
#================================================================
#      SEND/RECEIVE TEXT MESSAGES USING TCP/IP
#      =======================================
#
# This is the TCP/IP server program. It receives text messages
# from the keyboard and sends to an Android smart phone over
# a Wi-Fi link
#
# Author: Dogan Ibrahim
# File  : tcp2way.py
# Date  : December, 2023
#================================================================
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(("192.168.3.196", 5000))
sock.listen(1)

client, addr = sock.accept()                    # accept connection
print("Connected to client: ", addr)            # connected message

yn = 'y'

try:
   while yn == 'y':
      msg = input("Enter your message: ")       # read a message
      msg = msg +"\n"
      client.send(msg.encode('utf-8'))          # send the message

      msg = client.recv(1024)
      print("Received message: ")
      print(msg.decode('utf-8'))
```

```
        yn = input("Send more messages?: ")
        yn = yn.lower()

except KeyboardInterrupt:
        print("\nClosing connection to client")
        sock.close()
```

Figure 14.9 Program listing

Testing

You will be using the Android apps as in Figure 14.5. Start the Raspberry Pi 5 server program and then exchange messages between the smartphone and Raspberry Pi. Example communication is shown in Figure 14.10. In this example, Raspberry Pi sends message **Message from RASPBERRY PI**. In return, Android smartphone sends the message **message from ANDROID**.

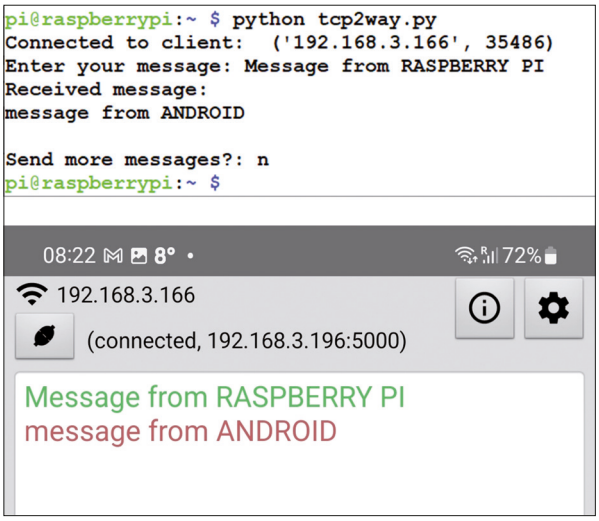

Figure 14.10 Example communication between Raspberry Pi 5 and Android apps

14.5 Project 3 – Communicating with a PC using TCP/IP

Description: In this project, a TCP/IP-based communication is established between the Raspberry Pi 5 and a PC running Python. Messages are exchanged between the Raspberry Pi 5 and the PC. The aim of this project is to show how TCP/IP communication can be established with a PC.

Background Information: In this project, the Raspberry Pi 5 is the server and the PC is the client. The programs on both sides are developed using the Python programming language. Python 3.10 is used on the PC. If you do not have Python on your PC, you could install it from the following website:

https://www.python.org/downloads/

Block diagram: Figure 14.11 shows the block diagram.

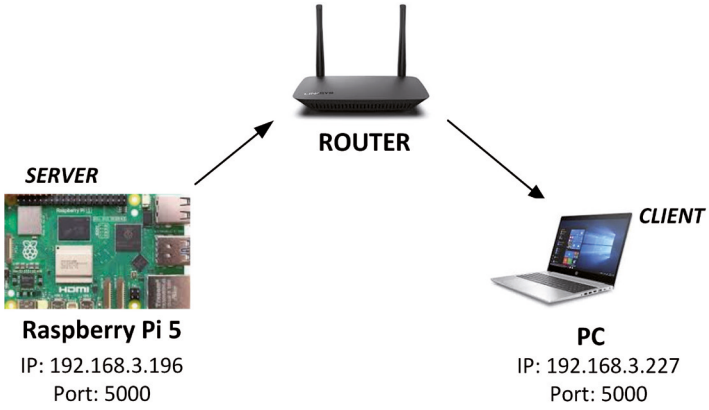

Figure 14.11 Block diagram

Raspberry Pi 5 program listing: The Raspberry Pi 5 program listing is shown in Figure 14.12 (**tcppc.py**). The program is very similar to the one given in Figure 14.9, i.e. program: **tcp2way.py**. You should terminate the program by entering **Ctrl+C**.

```
#================================================================
#      SEND/RECEIVE TEXT MESSAGES USING TCP/IP
#      =======================================
#
# This is the TCP/IP server program. It communicates with a PC
# running TCP/IP on the same port
#
# Author: Dogan Ibrahim
# File  : tcppc.py
# Date  : October, 2023
#================================================================
import socket
import time

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.bind(("192.168.3.196", 5000))
sock.listen(1)

client, addr = sock.accept()                    # accept connection
print("Connected to client: ", addr)            # connected message

try:
   while True:
      msg = input("Enter your message: ")       # read a message
```

```
      msg = msg +"\n"
      client.send(msg.encode('utf-8'))         # send the message

      msg = client.recv(1024)
      print("Received message: ", msg.decode('utf-8'))

except KeyboardInterrupt:
      print("\nClosing connection to client")
      sock.close()
      time.sleep(1)
```

Figure 14.12 Raspberry Pi 5 program listing

PC Program Listing: The PC program listing is shown in Figure 14.13 (**client.py**). The program creates a socket and connects to the server. Then, messages are exchanged between the client and the server.

```
#============================================================
#               TCP/IP CLIENT
#               =============
#
# This is the client program on the PC.The program exchanges
# messages with the server on the Raspberry Pi 5
#
# Author: Dogan Ibrahim
# File  : client.py
# Date  : October, 2023
#============================================================
import socket
import time
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
sock.connect(("192.168.3.196", 5000))

try:
    while True:
        msg = sock.recv(1024)
        print("Received message: ", msg.decode('utf-8'))
        data = input("Enter message to send: ")
        sock.send(data.encode('utf-8'))

except KeyboardInterrupt:
    print("Closing connection to server")
    sock.close()
    time.sleep(1)
```

Figure 14.13 PC program listing

The steps to run the program are as follows:

- Run the server program on the Raspberry Pi 5
- Run the client program on the PC
- Write messages as desired

Figure 14.14 shows a typical run of the two programs.

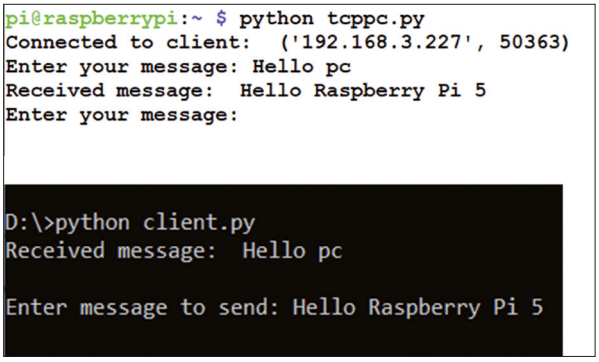

Figure 14.14 Example run of the program

Note: You may find that after exiting the program, you may not be able to run it again. This is because the socket stays open for about 30 seconds and the error message saying that the **Address is already in use** may be displayed. You can check the state of the port with the following command:

pi@raspberrypi:~ $ **netstat –n | grep 5000**

If the display includes the text **ESTABLISHED**, then it means that the socket has not been closed properly, and you will have to restart your Zero 2 W to run the program again. If on the other hand, you see the message with **TIME_WAIT**, then you should wait about 30 seconds before restarting the program.

14.6 Project 4 – Controlling an LED connected to Raspberry Pi 5 from a smartphone using TCP/IP

Description: In this project, an LED is connected to a Raspberry Pi 5. The LED is turned ON and OFF by sending commands ON and OFF respectively from an Android smartphone. The aim of this project is to show how an LED connected to a Raspberry Pi can be controlled from an Android smartphone remotely by sending commands using the TCP/IP protocol over a Wi-Fi link. In this project, Raspberry Pi 5 is the server, and the smartphone is the client.

Block diagram: Figure 14.15 shows the block diagram of the project.

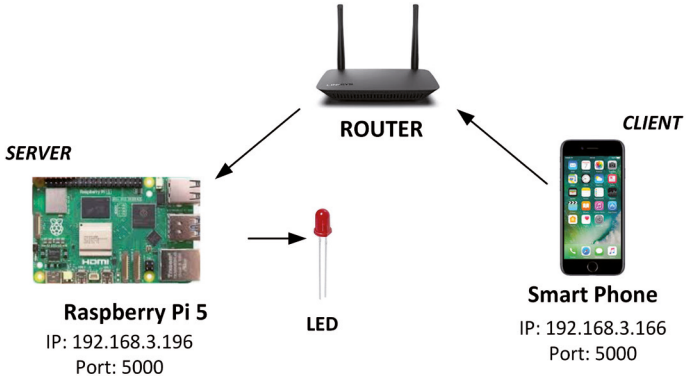

Figure 14.15 Block diagram of the project]#

The LED is connected to port pin GPIO 21 (pin 49) through a 470 Ohm current limiting resistor.

Program Listing: Figure 14.16 shows the program listing (program: **serverled.py**). As in the previous program, a socket is created and port 5000 is used. LED is assigned to port GPIO pin 21, and turned OFF at the beginning of the program. The server waits for a connection from the client and then accepts the connection and displays the message **Connected**. It then waits to receive a command from the client. If the command is **ON**, then the LED is turned ON. If, on the other hand, the command is **OFF**, then the LED is turned OFF. Sending command **X** terminates the server connection and exits the program.

```
#====================================================================
#              CONTROL LED FROM SMART PHONE
#              ============================
#
# In this program TCP/IP is used where Raspberry Pi 5 is the server
# and smart phone is the client. An LED connected to Raspberry Pi 5
# GPIO 21 and is controlled from the smart phone
#
# Author: DOgan Ibrahim
# File  : serverled.py
# Date  : December, 2023
#====================================================================
import socket
from gpiozero import LED
from time import sleep

led = LED(21)                          # LED at GPIO 21
led.off()

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

```
sock.bind(("192.168.3.196", 5000))       # Raspberry Pi 5 IP,port
sock.listen(1)
client, addr = sock.accept()
print("Connected")

data = [' '] * 10


while data != b'X\n':                    # Terminate?
  data = client.recv(1024)
  if data == b'ON\n':                    # ON
     led.on()
  elif data == b'OFF\n':                 # OFF
     led.off()

print("Closing connection")
GPIO.cleanup()
sock.close()
sleep(1)
```

Figure 14.16 Program listing

The program can be tested using the Android apps **TCP client** (Figure 14.5) used in Project 1. The server program started, then the client is started. Figure 14.17 shows sending the **ON** command to turn ON the LED.

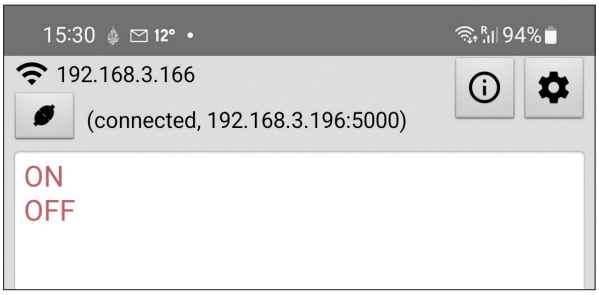

Figure 14.17 Command sent to turn ON/OFF the LED

Suggestions: The LED in this project can be replaced, for example with a relay and electrical equipment can be controlled remotely over Wi-Fi.

14.7 Project 5 – Sending a text message to a smartphone using UDP

Description: In this project, a UDP-based communication is established with an Android smartphone. The program reads text messages from the keyboard and sends to the smartphone. The aim of this project is to show how UDP communication can be established with an Android smartphone.

The block diagram is the same as in Figure 14.3.

Program Listing: In this project, Raspberry Pi 5 is the server and the smartphone is the client. Figure 14.18 shows the program listing (**udpserver.py**). At the beginning of the program, a UDP socket is created (**sock.SOCK_DGRAM**) and is then bind to port 5000. The server program then reads text messages sent from the smartphone and displays on the screen. Messages sent by Raspberry Pi 5 are displayed on the smartphone.

```
#============================================================
#      SEND TEXT MESSAGES USING UDP
#      ============================
#
# This is the UDP server program running on Raspberry Pi 5.
# The program exchanges text messages with an Android
# smart phone
#
# Author: Dogan Ibrahim
# File  : udpserver.py
# Date  : October, 2023
#============================================================
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("192.168.3.196", 5000))

try:
  while True:
    print()
    print("Waiting for messages")
    data, addr = sock.recvfrom(1024)
    print(addr)
    print("Received msg:", data.decode('utf-8'))
    msg = input("Message to send: ")
    sock.sendto(msg.encode('utf-8'), addr)
    print("Message sent")

except KeyboardInterrupt:
  print("\nClosing connection to client")
  sock.close()
```

Figure 14.18 Program listing

There are many UDP apps available free of charge for both Android and iOS smartphones. In this project, **UDP Sender/Receiver by JC Accounting & Innovative Technologies Inc** for Android smartphones is used (Figure 14.19).

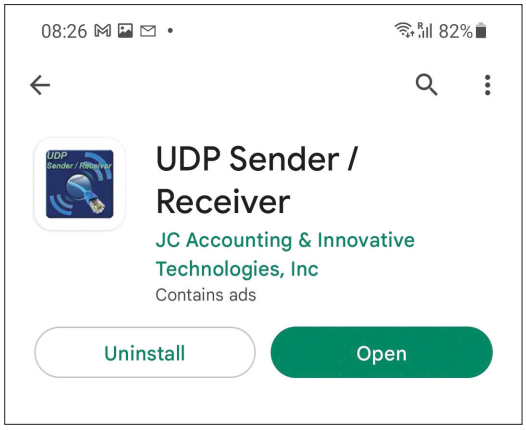

Figure 14.19 UDP Sender/Receiver apps

The steps to test the program are as follows:

- Start the server program on Raspberry Pi 5:

  pi@raspberrypi:~ $ **python udpserver.py**

- Start the smartphone apps and configure for the IP/Host and Port
- Write a message on mobile phone apps and click **SEND**. The message **Hello from Android** was sent as an example (Figure 14.20)
- Write a message on Raspberry Pi 5 and this message will be displayed on the smartphone. **Hello from Raspberry Pi 5** was sent for an example (Figure 14.20)
- Enter **Ctrl+C** on Raspberry Pi 5 to close the socket

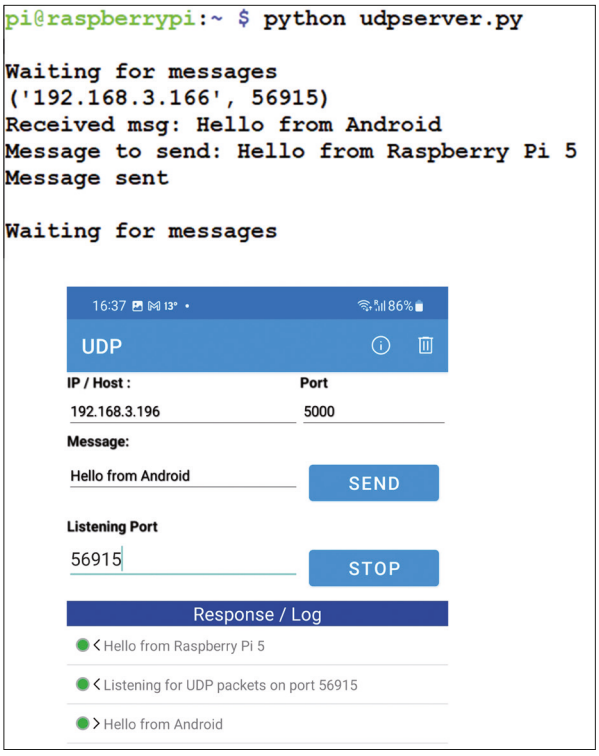

Figure 14.20 Sending and receiving messages

14.8 Project 6 – Controlling an LED connected to Raspberry Pi 5 from a smartphone using UDP

Description: In this project, an LED is connected to Raspberry Pi 5 port pin GPIO 21 (pin 40) through a 470 Ohm current limiting resistor. The LED is turned ON and OFF by sending commands **ON** and **OFF** respectively from an Android smartphone. The aim of this project is to show how an LED on the Raspberry Pi 5 can be controlled from a smartphone by sending commands using the UDP protocol over a Wi-Fi link. Here, the Raspberry Pi 5 is the server and the smartphone is a client.

The LED can easily be replaced with a relay, for example, to control electrical appliances from a smartphone.

Program Listing: Figure 14.21 shows the program listing (**udpled.py**). As in the previous program, a socket is created and the server waits to receive commands from a client to control the LED. If the command is **ON**, then the LED is turned ON. If on the other hand, the command is **OFF**, the LED is turned OFF. Command **X** terminates the server program.

```
#====================================================================
#      CONTROL LED FROM SMART PHONE
#      ============================
#
# In this program UDP is used where Zero 2 W is the server
# and smart phone is the client. An LED connected to the server
# and is controlled from the smart phone
#
# Author: DOgan Ibrahim
# File  : udpled.py
# Date  : October, 2023
#====================================================================
import socket
from gpiozero import LED
from time import sleep

led = LED(21)
led.off()

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("192.168.3.196", 2000))            # Bind to Zero 2 W IP,port

data = [' '] * 10
while data != b'X':
  data, addr = sock.recvfrom(1024)
  if data == b'ON':                           # ON command
     led.on()                                 # LED ON
  elif data == b'OFF':                        # OFF command
     led.off()                                # LED OFF

sock.close()
sleep(1)
```

Figure 14.21 Program listing

The program can be tested using the **UDP Sender/Receiver** apps used in Figure 14.19.

The steps to test the program are as follows:

- Construct the circuit on Raspberry Pi 5 with the LED
- Start the server program on Raspberry Pi 5:

  pi@raspberrypi:~ $ **python udpled.py**

- Start and configure the smartphone app

- Write the command **ON** and press **Send** on the smartphone (Figure 14.22). The LED should turn ON. Similarly, write **OFF** and the LED should turn OFF. Sending **X** should terminate the Raspberry Pi 5 program.

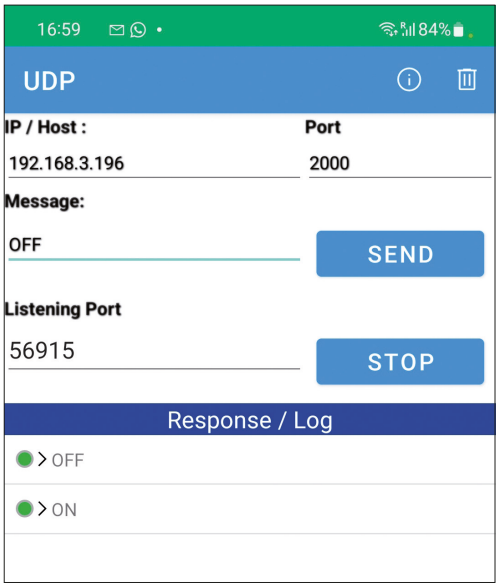

Figure 14.22 Turning ON/OFF the LED

14.9 Communicating with the Raspberry Pi Pico W over Wi-Fi

Raspberry Pi Pico W (it will be called Pico from now on) is a low-cost $6 microcontroller module based on the RP2040 microcontroller with dual-core Cortex-M0+ processor and with on-board Wi-Fi module. Figure 14.23 shows the front view of the Pico hardware module, which is basically a small board. At the middle of the board is the tiny 7 × 7 mm RP2040 microcontroller chip housed in a QFN-56 package. At the two edges of the board, there are 40 gold-coloured metal GPIO (General Purpose Input-Output) pins with holes. You should solder pins to these holes so that external connections can be made easily to the board. The holes are marked starting with number 1 in the top left corner of the board and the numbers increase downwards up to number 40 which is at the top right-hand corner of the board. The board is breadboard compatible (i.e. 0.1 inch pin spacing), and after soldering the pins, the board can be plugged into a breadboard for easy connection to the GPIO pins using jumper wires. Next to these holes, you will see bumpy circular cut-outs which can be plugged-in on top of other modules without having any physical pins fitted.

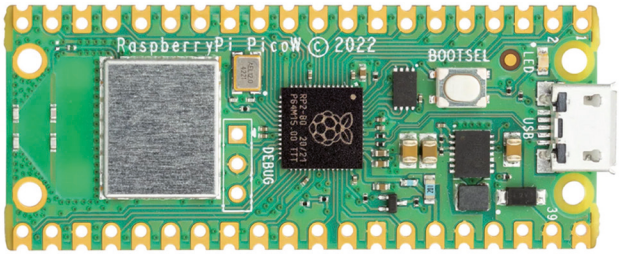

Figure 14.23 Front view of the Pico module

At one edge of the board, there is the micro-USB B port for providing power to the board and for programming the board. Next to the USB port, there is an on-board user LED that can be used during program development. Next to this LED there is a button named as BOOTSEL that is used during programming of the microcontroller as we will see in the next chapters. Next to the processor chip, there are 3 holes where external connections can be made to. These are used to debug your programs using Serial Wire Debug (SWD). At the other edge of the board is the single-band 2.4 GHz Wi-Fi module (802.11n). Next to the Wi-Fi module is the on-board antenna.

You will notice the following types of letters and numbers at the back of the board:

GND	-	power supply ground (digital ground)
AGND	-	power supply ground (analog ground)
3V3	-	+3.3 V power supply (output)
GP0 – GP22	-	digital GPIO
GP26_A0 – GP28_A2	-	analog inputs
ADC_VREF	-	ADC reference voltage
TP1 – TP6	-	test points
SWDIO, GND, SWCLK	-	debug interface
RUN	-	default RUN pin. Connect LOW to reset the RP2040
3V3_EN	-	this pin by default enables the +3.3 V power supply. +3.3 V can be disabled by connecting this pin LOW
VSYS	-	system input voltage (1.8V to 5.5V) used by the on-board SMPS to generate +3.3 V supply for the board
VBUS	-	micro-USB input voltage (+5V)

Some of the GPIO pins are used for internal board functions. These are:

GP29 (input)	-	used in ADC mode (ADC3) to measure VSYS/3
GP24 (input)	-	VBUS sense - HIGH if VBUS is present, else LOW
GP23 (output)	-	Controls the on-board SMPS Power Save pin

The specifications of the Pico hardware module are as follows:

- 32-bit RP2040 Cortex-M0+ dual-core processor operating at 133 MHz
- 2 MB Q-SPI Flash memory
- 264 KB SRAM memory
- 26 GPIO (+3.3 V compatible)
- 3× 12-bit ADC pins
- Accelerated floating-point libraries on chip
- On-board single-band Infineon CYW43439 wireless chip, 2.4 GHz wireless interface (802.11b/g/n) and Bluetooth 5.2
- Serial Wire Debug (SWD) port
- Micro-USB port (USB 1.1) for power (+5 V) and data (programming)
- 2× UART, 2× I²C, 2× SPI bus interface
- 16× PWM channels
- 1× Timer (with 4 alarms), 1× Real-Time Clock
- On-board temperature sensor
- On-board LED at GPIO0, controlled by the CYW43439 chip
- Castellated module, allowing soldering direct to carrier boards
- 8× Programmable IO (PIO) state machines for custom peripheral support
- MicroPython, C, C++ programming
- Drag & drop programming using mass storage over USB

Pico GPIO hardware is +3.3 V compatible, and it is therefore important to be careful not to exceed this voltage when interfacing external input devices to the GPIO pins. +5 V to +3.3 V logic converter circuits or resistive potential divider circuits must be used if it is required to interface devices with +5 V outputs to the Pico GPIO pins.

Pico can be programmed using MicroPython or C/C++ languages. It is assumed that the readers have Pico development boards with the MicroPython installed. It will also be useful if the readers are familiar using Thonny with the Pico. A book entitled **Raspberry Pi Pico W** and written by the author is available from Elektor and interested readers might consider purchasing this book for developing Pico-based projects.

Figure 14.24 shows the pin configuration of the Pico.

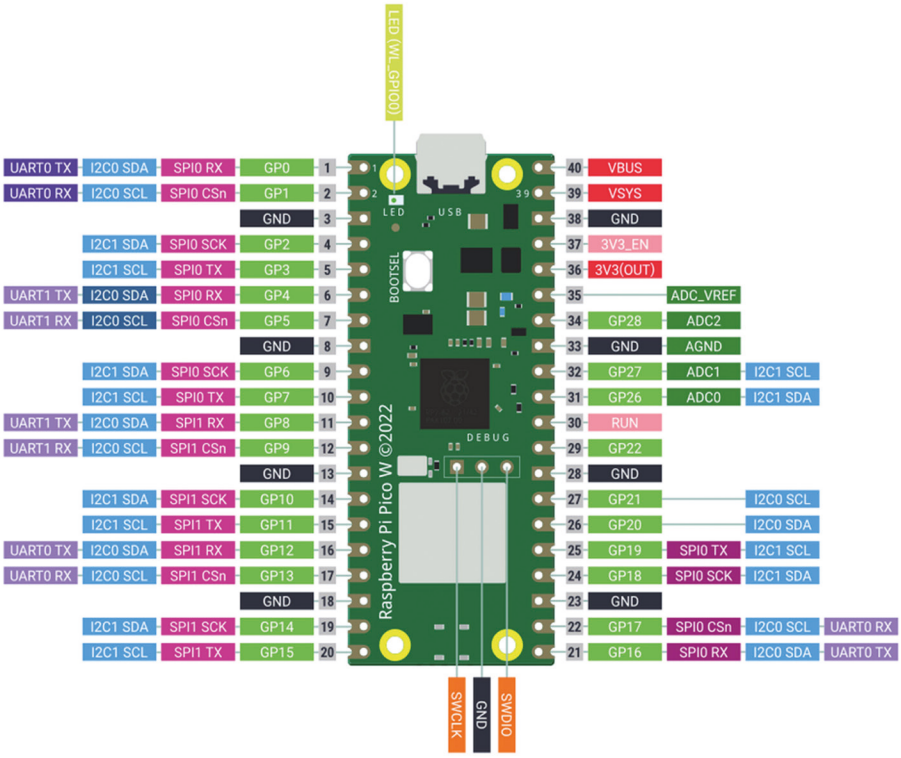

Figure 14.24 Pico pin configuration

14.9.1 Project 7 – Raspberry Pi 5 and Raspberry Pi Pico W communication – controlling a relay over Wi-Fi

Description: In this project, you have a Raspberry Pi 5 and Raspberry Pi Pico W. A push-button is connected to Pico, and a +3.3 V relay is connected to the Raspberry Pi 5. Pressing the button on the Pico sends a command to the Raspberry Pi over the Wi-Fi to activate the relay. The relay remains active for 5 seconds. In this project, Raspberry Pi and Pico communicate using the UDP protocol. Raspberry Pi is the server and Pico is the client.

Block diagram: Figure 14.25 shows the block diagram of the project.

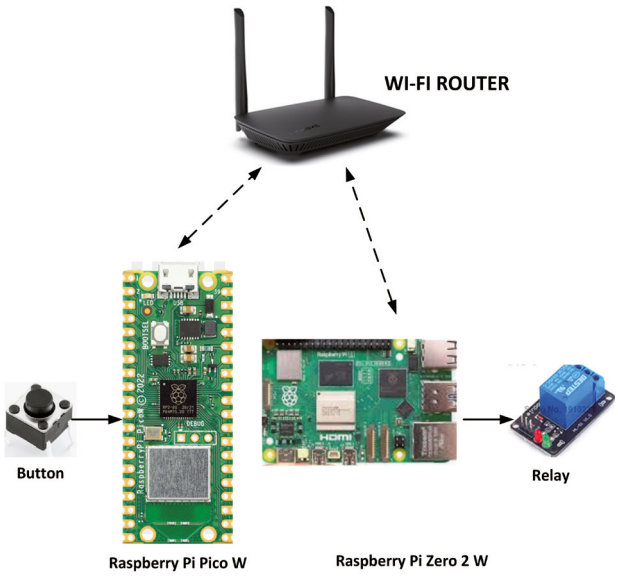

Figure 14.25 Block diagram of the project

Circuit diagram: The circuit diagram of the project is shown in Figure 14.26 with the button and relay connected to the Pico and Raspberry Pi respectively.

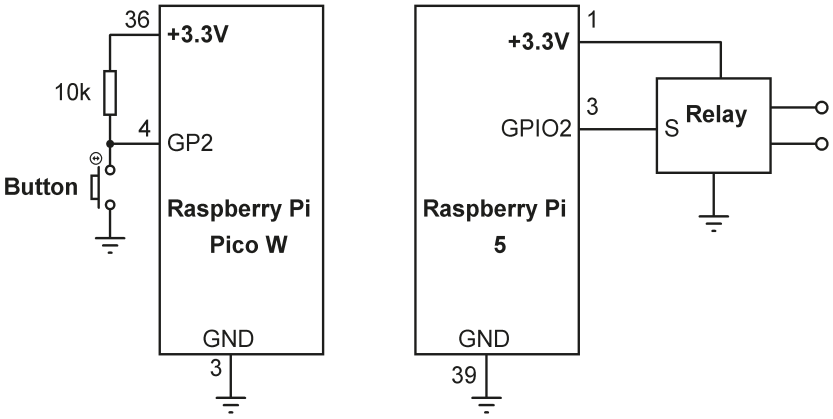

Figure 14.26 Circuit diagram of the project

Pico program listing: Figure 14.27 shows the Pico program listing (**picoudp.py**). At the beginning of the program, LED is assigned to port GP2 and is turned OFF. The Function **Connect()** is called to connect to the local Wi-Fi network. Then, a socket is created with port number 2000 and IP address 192.168.3.21. When the Button is pressed, the program sends 1 to the Raspberry Pi 5 so that the LED can be turned ON. This process is repeated after a 1-second delay.

```
#-------------------------------------------------------------
#      RASPBERRY PI PICO W - RASPBERRY PI 5 COMMS
#      ==========================================
#
# In this project a pushbutton is connected to GP2 of PICO W.
# Presingthe button sends a command to Raspberry Pi 5 to
# activate a relay. UDP protocol is used in this project
#
# Author: Dogan Ibrahim
# File  : picoudp.py
# Date  : October, 2023
#-------------------------------------------------------------
from machine import Pin
import network
import socket
import utime
global wlan

BUTTON = Pin(2, Pin.IN)                                 # Button at GP2

#
# This function attempts to connect to Wi-Fi
#
def connect():
    global wlan
    wlan = network.WLAN(network.STA_IF)
    while wlan.isconnected() == False:
       print("Waiting to be connected")
       wlan.active(True)
       wlan.connect("TP-Link_6138_EXT", "24844604")
       utime.sleep(5)

connect()
print("Connected")
UDP_PORT = 2000                                         # Port used
UDP_IP = "192.168.3.21"                                 # Zero 2W IP
cmd = b"1"                                              # Cmd to turn ON
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

while True:
    while BUTTON.value() == 1:                          # Not pressed
       pass
    while BUTTON.value() == 0:                          # Not released
       pass
    sock.sendto(cmd, (UDP_IP, UDP_PORT))                # Send cmd
    print("Command sent")                               # Message
    utime.sleep(1)                                       Wait 1 sec
```

Figure 14.27 Raspberry Pi Pico W program listing (picoudp.py)

Raspberry Pi 5 program listing: Figure 14.28 shows the Raspberry Pi 5 program listing (**RPiudp.py**). At the beginning of the program the libraries used are imported, the relay control output is configured at port GPIO 2 and is deactivated. Then a socket is created, and the program binds to it with the Raspberry Pi 5 address. The program then waits to receive a command from the Pico. The received command is stored in variable **data** and if it is 1, then the relay is activated for 5 seconds. At the end of this time, the relay is deactivated and the program repeats waiting for a command.

```
#============================================================
#      RASPBERRY PI PICO W - RASPBERRY PI 5 COMMS
#      ==========================================
#
# This is the UDP server program running on Raspberry Pi 5.
# The program receives a command from PICO W and activates a
# relay connected to GPIO 2 for 5 seconds
#
# Author: Dogan Ibrahim
# File  : RPiudp.py
# Date  : October, 2023
#============================================================
from gpiozero import LED
import socket
from time import sleep

RELAY = LED(2)                          # Relay at port GPIO 2
RELAY.off()                             # RELAY off

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind(("192.168.3.196", 2000))

try:
  while True:
    data, addr = sock.recvfrom(1024)    # GEt command
    if data == b'1':                    # Command is 1?
       RELAY.on()                       # Activate Relay
       sleep(5)                         # 5 seconds delay
       RELAY.off()                      # Deactivate Relay

except KeyboardInterrupt:               # Keyboard interrupt
  print("\nClosing connection to client")
  sock.close()
```

Figure 14.28 Raspberry Pi 5 program listing (RPiudp.py)

Testing the project
The steps to test the project are:

- Run the server on Raspberry Pi 5:

  pi@raspberry: ~$ **python RPiudp.py**

- Run the Pico program in Thonny by clicking the green Run button. You should see the message **Connected** when Pico connects to the local router.

- Push the button on Pico. The message **Command** sent will be displayed on Pico terminal. A packet will be sent to Raspberry Pi 5 which will turn ON the LED for 5 seconds

- Enter **Ctrl+C** to terminate the program

14.10 Project 8 - Storing ambient temperature and atmospheric pressure data on the Cloud

Description: In this project, the ambient temperature and atmospheric pressure are read and stored on the Cloud. A BME280 type sensor module (see Chapter 10.6) is used in this project.

Block diagram: The block diagram of the project is shown in Figure 14.29.

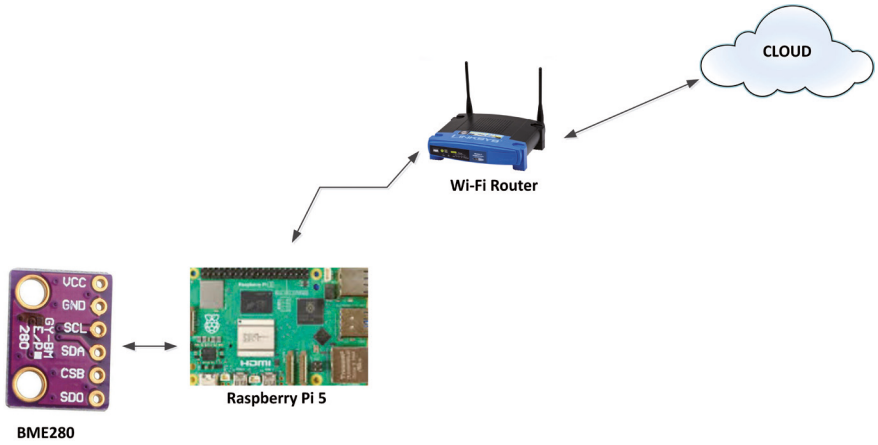

Figure 14.29 Block diagram of the project

Circuit diagram: Figure 14.30 shows the circuit diagram. SCL and SDA pins of BME280 are connected to SDA (pin 3) and SCL (pin 5) of Raspberry Pi 5. The sensor is powered from +3.3 V.

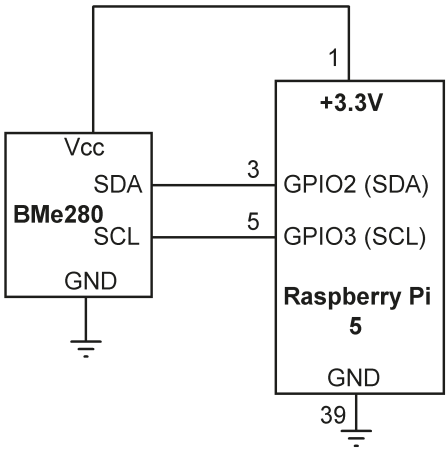

Figure 14.30 Circuit diagram of the project

The Cloud

There are several cloud services that can be used to store data (for example **SparkFun**, **ThingSpeak, Cloudino, Bluemix** etc.). In this project, the **Thingspeak** is used. This is a free cloud service where sensor data can be stored and retrieved using simple HTTP requests. Before using the Thingspeak we have to create an account on their website and then log in to this account.

Go to the ThingSpeak website:

https://thingspeak.com/

*Click **Get Started For Free** and create an account if you don't already have one*. Then, you should create a New Channel by clicking on **New Channel**. Fill in the form as shown in Figure 14.31. Give the name **Raspberry Pi 5** to the application, give a description, and create two **fields** called **Atmospheric Pressure** and **Temperature**. You can optionally fill in the other items as well if you wish.

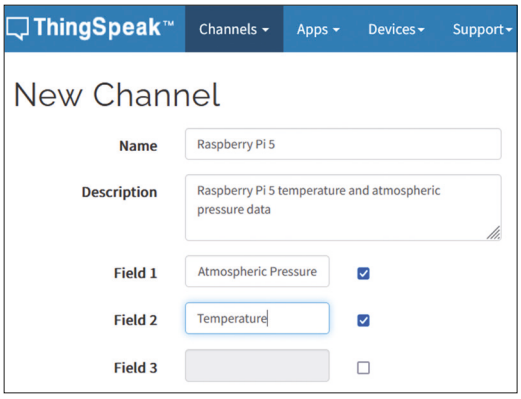

Figure 14.31 Create a New Channel (only part of the form shown)

Click **Save Channel** at the bottom of the form. Your channel is now ready to be used with your data. You will now see tabs with the following names. You can click on these tabs and see the contents to make corrections if necessary:

- Private View: This tab displays private information about your channel where only you can see.
- Public View: If your channel is public, use this tab to display selected fields and channel visualizations.
- Channel Settings: This tab shows all the channel options you set at creation. You can edit, clear, or delete the channel from this tab.
- API Keys: This tab displays your channel API keys. Use the keys to read from and write to your channel.
- Data Import/Export: This tab enables you to import and export channel data.

You should click the **API Keys** tab and save your unique **Write API** and **Read API** keys and unique **Channel ID** in a safe place, as you will need to use them in our program. The API Keys and the Channel ID in this project were as in Figure 14.32.

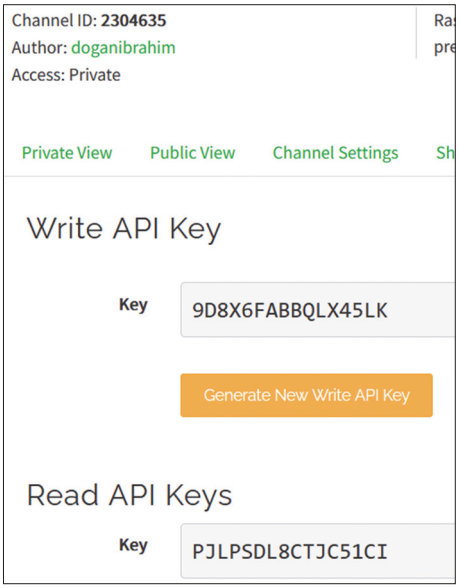

Figure 14.32 Author's Channel ID and API Keys

Also, select the **Public View** and navigate to **Sharing**. You may select the option **Share channel view with everyone** so that everyone can access your data remotely.

Program listing: In this program you will be using the BME280 library as in Chapter 10.6. The steps to install the library are not repeated here.

After constructing the circuit, you should check to make sure that the BME280 is detected by the Raspberry Pi 5. Enter the following command:

```
pi@raspberrypi:~ $ sudo i2cdetect –y 1
```

You should see the hardware address of the BME280 chip displayed as 76 (see Figure 14.33).

```
pi@raspberrypi:~ $ sudo i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:                         -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- 76 --
pi@raspberrypi:~ $
```

Figure 14.33 BME280 hardware address detected

Figure 14.34 shows the program listing (**Cloud.py**). At the beginning of the program, the libraries used are imported to the program. Thingspeak **Write Key** and **Host Address** are defined. The main program loop starts with the **while** statement. Inside this loop, the IP address of the **Thingspeak** website is extracted, and a connection is made to this site at port 80. Then the atmospheric pressure and temperature readings are obtained from the BMP280 module and are included in the **path** statement. The **sock.send** statement sends an HTTP GET request to the **ThingSpeak** site and uploads the pressure and temperature values. This process is repeated every 30 seconds.

Figure 14.35 shows the pressure and temperature data plotted by **ThingSpeak**. The Chart Options can be clicked to change various parameters of the charts. For example, Figure 14.36 shows the temperature as a column display. In Figure 14.37 the pressure is shown as a step graph. A title and X-axis label is added in Figure 14.38 to the pressure graph. Figure 14.39 shows the current temperature displayed in a clock format (click **Add Widgets** for this type of display). Figure 14.40 shows the current temperature in digital format.

Because the Channel was saved as Public, you can view the graph from a web browser (see Figure 14.41) by entering the Channel ID. In this project, the link to view the data graphs from a web browser is:

https://api.thingspeak.com/channels/2304635

We can also export some or all of the fields in CSV format by clicking **Export recent data** so that it can be analysed by external statistical packages such as Excel.

```
#------------------------------------------------------------------------------
#        ATMOSPHERIC PRESSURE AND TEMPERATURE ON THE CLOUD
#       ================================================
#
# The ambient temperature and pressure sensor BMPE280 is connected to Raspberry
# Pi 5.The project reads the temperature and atmospheric pressure and sends
```

```
# to the Cloud where it can be accessed from anywhere. In addition, change
# of the temperature and the pressure can be plotted in the cloud.
#
#
# The program uses the Thingspeak cloud service
#
# Author: Dogan Ibrahim
# File  : Cloud.py
# Date  : October, 2023
#-------------------------------------------------------------------------------
import socket
from time import sleep
from bme280pi import Sensor

sensor = Sensor(address = 0x76)

APIKEY = "9D8X6FABBQLX45LK"                    # Thingspeak API key
host = "api.thingspeak.com"                    # Thigspeak host

#
# Send data to Thingspeak. This function sends the temperature and
# humidity data to the cloud every 30 seconds
#
while True:
   sock = socket.socket()
   addr = socket.getaddrinfo("api.thingspeak.com",80)[0][-1]
   sock.connect(addr)
   data = sensor.get_data()
   p = data['pressure']                                # Pressure in haP
   t = data['temperature']                     # Temperature in C
   path = "api_key="+APIKEY+"&field1="+str(p)+"&field2="+str(t)
   sock.send(bytes("GET /update?%s HTTP/1.0\r\nHost: %s\r\n\r\n"
%(path,host),"utf8"))
   sleep(5)
   sock.close()
   sleep(25)
```

Figure 14.34 Program listing

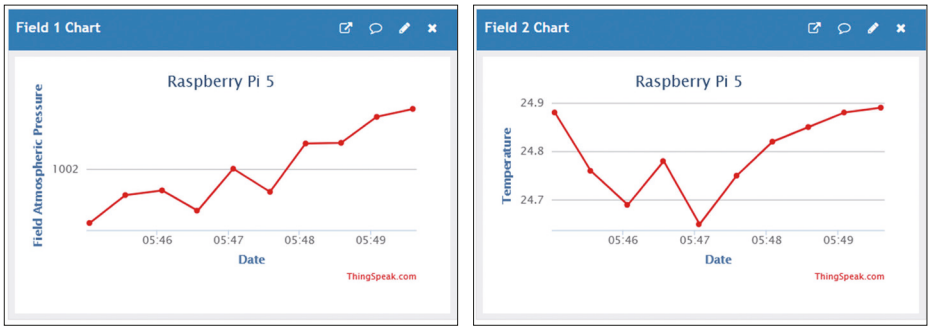

Figure 14.35 Plotting the pressure and temperature

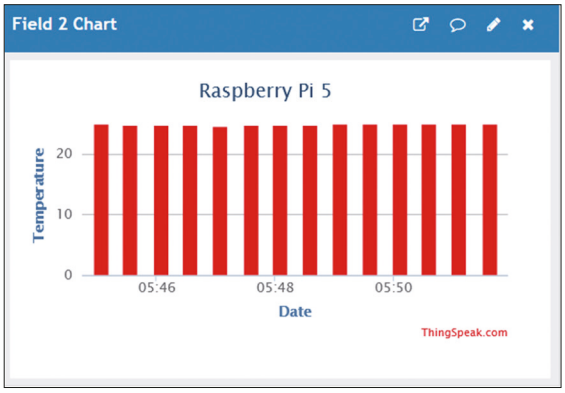

Figure 14.36 Displaying temperature as columns

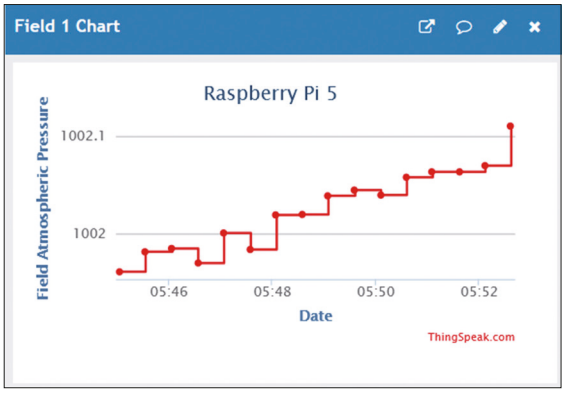

Figure 14.37 Displaying pressure as steps

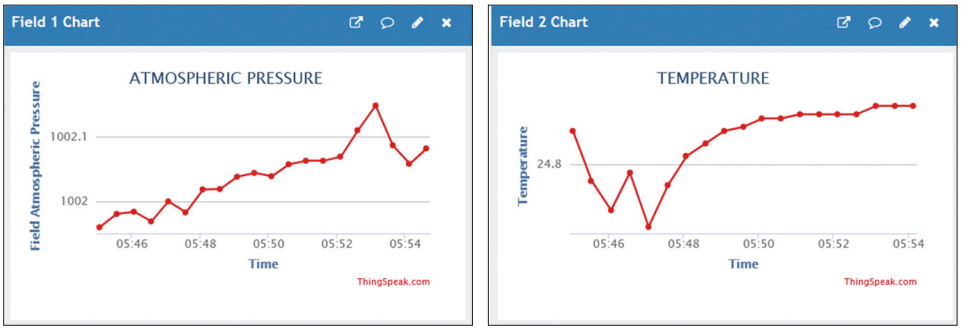

Figure 14.38 Adding title and x-axis label

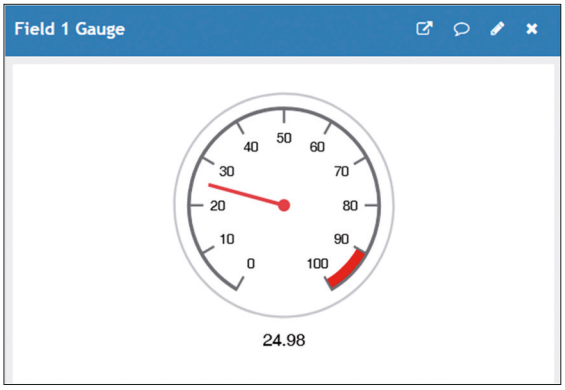

Figure 14.39 Displaying the current temperature in a clock format

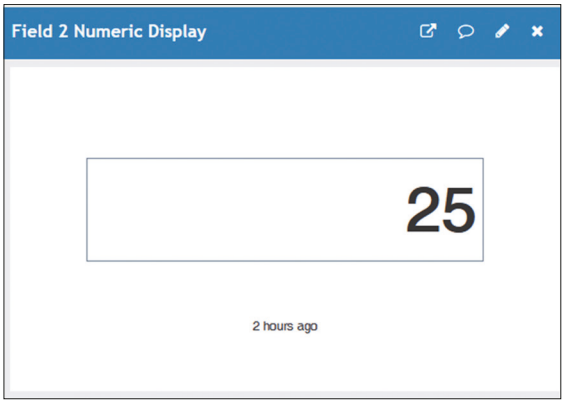

Figure 14.40 Displaying the current temperature in digital format

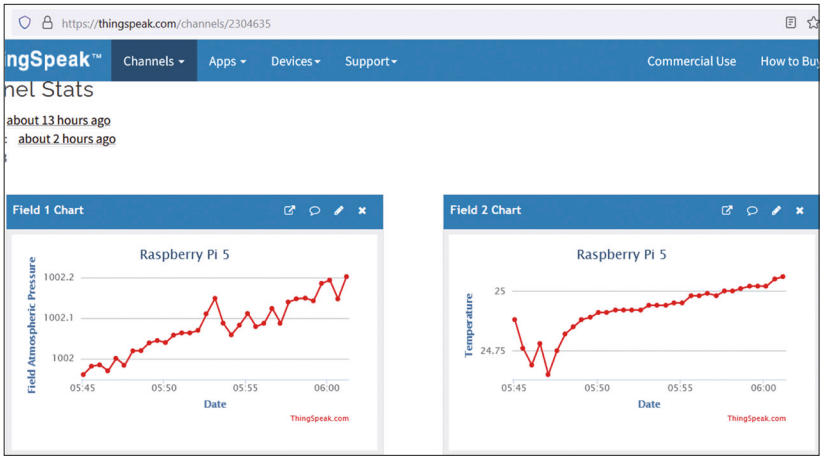

Figure 14.41 Displaying the graphs from a website

Chapter 15 • Communication over Bluetooth

15.1 Overview

In the last chapter, you have learned how to write programs to use the Wi-Fi and communicate with other devices over the LAN using the UDP and TCP protocols. In this chapter, you will develop programs for using the Bluetooth communication.

Bluetooth is a short-range communication technology and is commonly used to communicate with other devices, mainly developed with the IoT applications in mind. All smartphones nowadays support communication through Bluetooth. Bluetooth operates at 2.4 GHz, with data rates lower than that of Wi-Fi. Bluetooth is not as secure as Wi-Fi, but it is easier to use. The power consumption of Bluetooth is lower compared to Wi-Fi, and also it has a shorter range than Wi-Fi. Bluetooth is a packet-based protocol with a master-slave architecture, where one master may communicate with up to seven slaves. The effective range of Bluetooth depends on propagation conditions, antenna configuration, power supply condition, material coverage and so on. Most Bluetooth applications are for indoors where because of the walls the signals attenuate and result in shorter range.

Raspberry Pi 5 supports both the Classic Bluetooth and Bluetooth Low Energy (BLE). BLE is intended for reduced power applications while providing reasonable range. Most smartphones, including Android, iOS, Windows Phone, Blackberry, macOS and many others support the BLE. BLE uses the same 2.4 GHz radio frequency as the classic Bluetooth and dual-mode devices, therefore can share the same single antenna. Classical Bluetooth can handle large amounts of data quickly, whereas BLE has been developed to handle smaller amounts of data.

15.2 Project 1 – Exchanging text with a smartphone

Description: In this project, Classical Bluetooth communication is established between the Raspberry Pi 5 and an Android smartphone. Text messages are exchanged between the two devices.

Block diagram: Figure 15.1 shows the block diagram of the project.

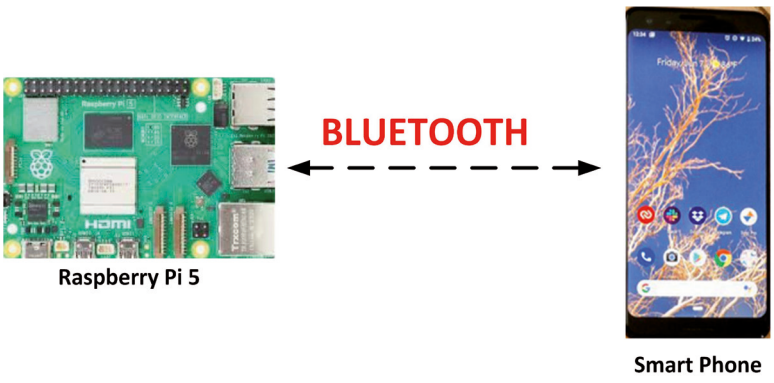

Figure 15.1 Block diagram of the project

Enabling Bluetooth

Before using your smartphone for Bluetooth applications, you must enable the Bluetooth on it. Depending on the model of the smartphone you have, this is usually done from the **Settings** menu.

Similarly, before using Bluetooth on your Raspberry Pi, you must enable it. There are two ways you can enable Bluetooth on the Raspberry Pi 5: using the graphical desktop (GUI mode), or using the Console mode.

Using the Graphical Desktop

The steps for enabling Bluetooth on the Raspberry Pi 5 using the graphical desktop are given below:

- Enable Bluetooth on your smartphone
- If you have a monitor connected directly to the Raspberry Pi 5 then skip the next line
- Start the VNC server on your Raspberry Pi 5 and login using the VNC Viewer.
- Click on the blue Bluetooth icon on your Raspberry Pi 5 screen at the top right-hand side, and turn Bluetooth ON, if it is not already ON. Then, select **Make Discoverable.** You should see the Bluetooth icon flashing. Click **Add Device**
- Select **raspberrypi** in the Bluetooth menu (**raspberrypi** is the default Bluetooth name of your Raspberry Pi 5) on your mobile device (you may have to scan on your mobile device). You should see the **Connecting** message on your smart device.
- Click **Pair** to accept the pairing request on your Raspberry Pi 5 as shown in Figure 15.2

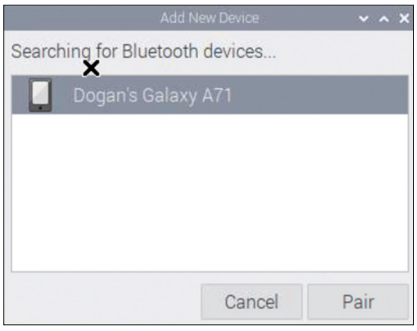

Figure 15.2 Bluetooth pairing request on Raspberry Pi 5

- You should now see the message **Pairing Successfully** on your Raspberry Pi 5

Using Console mode

You can enable Bluetooth on your Raspberry Pi 5 using the Console mode. Additionally, you can make Bluetooth discoverable, scan for nearby Bluetooth devices and then connect to a Bluetooth device. The steps are given below (characters typed by the user are in bold for clarity):

- Find the Bluetooth MAC address of your smartphone. For Android phones, the steps are usually:
- Go to the **Settings** menu
- Tap **About Phone**
- Tap **Status information**
- Scroll down to see your **Bluetooth address** (e.g. Figure 15.3). In this example, the MAC address was **50:50:A4:0F:62:3F**

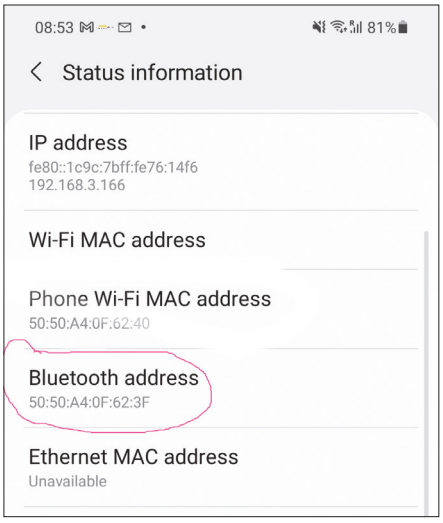

Figure 15.3 Bluetooth MAC address

- Make your Bluetooth discoverable with the following command:

  pi@raspberrypi: ~ $ **sudo hciconfig hci0 piscan**

- Start the Bluetooth tool on your Raspberry Pi 5 from the command mode:

  pi@raspberrypi:~ $ **bluetoothctl**

- Turn Bluetooth ON:

  [bluetooth]# **power on**

- Configure Bluetooth to run:

  [bluetooth]# **agent on**
  [bluetooth]# **default-agent**

- Make device discoverable:

  [bluetooth]# **discoverable on**

- Scan for nearby Bluetooth devices, you may have to wait several minutes:

  [bluetooth]# **scan on**

- Enter command **devices** to see the nearby Bluetooth devices (see Figure 15.4). You may have to wait a couple of minutes for the display to update. Make a note of the MAC address of the device you wish to connect to (Android mobile phone in this project) as we will be using this address to connect to the device. An example is shown in Figure 15.4:

  [Bluetooth]# **devices**

```
[NEW] Device 69:FC:9E:62:DA:4D 69-FC-9E-62-DA-4D
[DEL] Device 69:FC:9E:62:DA:4D 69-FC-9E-62-DA-4D
[NEW] Device 69:FC:9E:62:DA:4D 69-FC-9E-62-DA-4D
[DEL] Device 69:FC:9E:62:DA:4D 69-FC-9E-62-DA-4D
[NEW] Device 50:50:A4:0F:62:3F Dogan's Galaxy A71
[CHG] Device 50:50:A4:0F:62:3F RSSI: -71
```

Figure 15.4 Nearby Bluetooth devices

  In this example, the author's smartphone is **Galaxy A71** and the Bluetooth MAC address is: **50:50:A4:0F:62:3F**

- Pair the device:

  [bluetooth]# **pair 50:50:A4:0F:62:3F**

- Connect to our smartphone:

  [bluetooth]# **connect 50:50:A4:0F:62:3F**

- Enter **yes** to confirm **passkey**

- Accept pairing on your smartphone

- You should see message **device 50:50:A4:0F:62:3F Connected : yes** displayed

- Exit from the Bluetooth tool by entering **Ctrl+Z**

You can find the Bluetooth MAC address of your Raspberry Pi 5 by entering the following command:

```
pi@raspberrypi:~ $ hciconfig | grep "BD Address"
```

You can change the Bluetooth broadcast name by the following command:

```
pi@raspberrypi:~ $ sudo hciconfig hci0 name "new name"
```

To see your Bluetooth broadcast name, enter:

```
pi@raspberrypi:~ $ sudo hciconfig hci0 name
```

Some other useful Raspberry Pi 5 Bluetooth commands are:

- To reset the Bluetooth adapter: **sudo hciconfig hci0 reset**
- To restart Bluetooth: **sudo invoke-rc.d bluetooth restart**
- To list Bluetooth adapters: **hciconfig**

Python Classical Bluetooth Library
You will need to install the Python Classical Bluetooth library before developing your program. This is done by entering the following command in the command mode:

```
pi@raspberrypi:~ $ sudo apt-get install bluez python3-bluez
```

Accessing From the Mobile Phone
To access the Raspberry Pi 5 from a smartphone app, make the following changes to your Raspberry Pi 5 from the command line:

- Start **nano** to edit the following file:

  ```
  pi@raspberrypi:~ $ sudo nano /etc/systemd/system/dbus-org.
  bluez.service
  ```

- Add **–C** at the end of the **ExecStart=** line. Also add another line after the ExecStart line. The final two lines should look like:

  ```
  ExecStart=/usr/libexec/bluetooth/bluetoothd -C
  ExecStartPost=/usr/bin/sdptool add SP
  ```

- Exit and save the file by entering **Ctrl+X** followed by **Y**

- Reboot the Raspberry Pi 5:

```
pi@raspberrypi:~ $ sudo reboot
```

Program listing: Figure 15.5 shows the program listing (**bluetxt.py**). Do not call your program **Bluetooth.py**! The Bluetooth code is similar to TCP/IP code. At the beginning of the program, modules socket, and Bluetooth are imported to the program. The program then creates a Bluetooth socket, binds and listens on this socket, and then waits to accept a connection. The remainder of the program is executed in a loop, where the program issues the statement **ClientSock.recv** and waits to read data from the smartphone. The received data is decoded and displayed on the screen. The user is then expected to send a text message to the smartphone. This message is displayed on the smartphone's screen. This process is repeated until stopped by the user.

```
#=======================================================
#              BLUETOOTH COMMUNICATION
#              =======================
#
# In this project text messages are exchanged with a smart
# phone using the Bluetooth protocol
#
# Author: Dogan Ibrahim
# File  : bluetxt.py
# Date  : October, 2023
#=======================================================
import socket
import bluetooth

#
# Start of main program loop.Configure Bluetooth, create a
# port, listen for client connections, and accept connection
#
port = 1
ServerSock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
ServerSock.bind(("", port))
ServerSock.listen(1)
ClientSock, addr = ServerSock.accept()

#
# Now receive text from smart phone and display
#
try:

   while True:
      data = ClientSock.recv(1024)                    # receive text
      print("Received data: ", data.decode('utf-8'))
```

```
        msg = input("Enter data to send: ")              # TExt to send
        ClientSock.send(msg.encode('utf-8'))             # Send text

except KeyboardInterrupt:                                # Keyboard int
        ServerSock.close()                               # Close socket
```

Figure 15.5 Program listing

Testing
The project can be tested by the following steps:

- Make sure that Bluetooth is enabled on both the smartphone and Raspberry Pi 5 and they are paired
- You can use the freely available Bluetooth apps on our smartphone to communicate with the Raspberry Pi 5. In this project, the apps called **Bluetooth Terminal HC05 by mightyIT** is used (Figure 15.6)

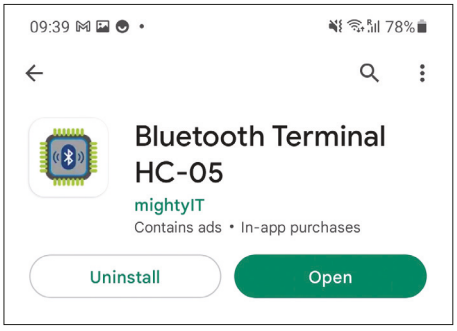

Figure 15.6 Android Bluetooth apps used in the project

- Start the Raspberry Pi program:

  pi@raspberrypi:~ $ **python bluetxt.py**

- Start the smartphone apps and select the paired Raspberry Pi 5 device (e.g. **raspberrypi**)
- Enter a message in column **Enter ASCII Command** and press **Send ASCII**. The message will be sent to the Raspberry Pi 5.
- Enter a message on the Raspberry Pi 5. This message will be sent and displayed on the smartphone top part of the screen
- Figure 15.7 shows an example message exchange between the Raspberry Pi 5 and the smartphone. In this example, the smartphone sends the message: **message from the smartphone** and Raspberry Pi 5 sends the message:

message from Raspberry Pi 5

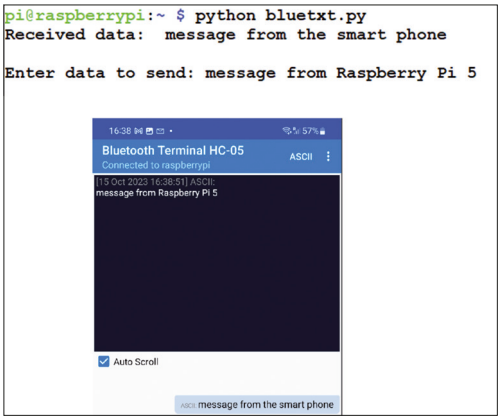

Figure 15.7 Message exchange between the Raspberry Pi 5 and a smartphone

- Enter **Ctrl+C** to close the socket and terminate the program

15.3 Project 2 – Bluetooth control of LED from a smartphone

Description: In this project, an LED is connected to port GPIO 21 (pin 40) of Raspberry Pi 5 through a 470 Ohm current limiting resistor. The LED is controlled by sending commands from an Android smartphone using Bluetooth communication.

The following commands can be sent from the Android smartphone to control the LED:

L1	Turn the LED ON
L0	Turn the LED OFF

Block diagram: Figure 15.8 shows the block diagram of the project.

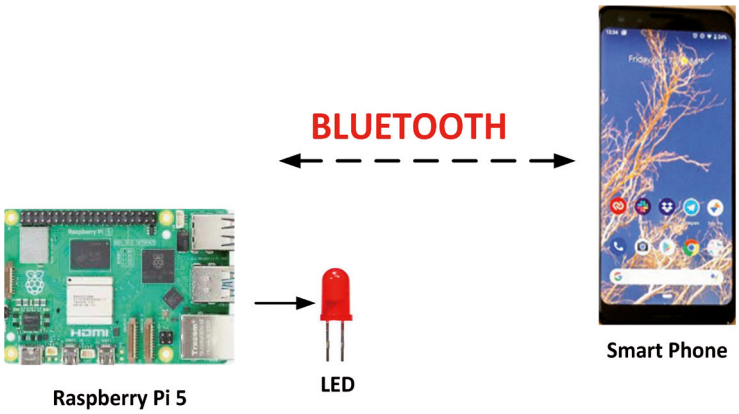

Figure 15.8 Block diagram of the project

Program Listing: Figure 15.9 shows the program listing of the project (**blueled.py,** do not call your program **Bluetooth.py**!). The Bluetooth code is similar to TCP/IP code. The LED port is defined and configured as output through the **gpiozero** module. The program then creates a Bluetooth socket, binds and listens on this socket, and then waits to accept a connection. The remainder of the program is executed in a loop, where the program issues the statement **ClientSock.recv** and waits to read data from the smartphone. Note that the smartphone app automatically appends carriage-return and line-feed characters to the end of the data (i.e. **\r\n**)

```
#=======================================================
#              LED CONTROL BY BLUETOOTH
#              ========================
#
# In this project an LED is connected to GPIO 21.The LED
# LED is controlled by sending commands from an Android
# smart phone using a Bluetooth apps.
#
# Valid commands are:
# L1 Turn ON the LED
# L0 Turn OFF the LED
#
# Author: Dogan Ibrahim
# File  : blueled.py
# Date  : October, 2023
#=======================================================
import socket
import bluetooth
from gpiozero import LED

#
# LED is at GPIO 21, configure as output and turn OFF
#
led = LED(21)                                  # LED at port 21
led.off()                                      # LED off

#
# Start of main program loop.Configure Bluetooth, create a
# port, listen for client connections, and accept connection
#
port = 1
ServerSock = bluetooth.BluetoothSocket(bluetooth.RFCOMM)
ServerSock.bind(("", port))
ServerSock.listen(1)
ClientSock, addr = ServerSock.accept()

#
```

```
# Now receive comamnds and decode
#
try:

   while True:
     data = ClientSock.recv(1024)              # receive command
     if data == b'L1\r\n':                     # L1?
         led.on()                              # turn ON LED
     elif data == b'L0\r\n':                   # L0?
          led.off()                            # turn OFF LED

except KeyboardInterrupt:                      # Interrupt
     ServerSock.close()
```

Figure 15.9 Program listing

Testing
The same Android app as in the previous project is used. The steps are:

- Make sure that the Bluetooth is enabled on both the smartphone and Raspberry Pi 5 and that they are already paired
- Start the Raspberry Pi 5 program:

  pi@raspberrypi:~ $ **python blueled.py**

- Start the apps as before. To send command **L1**, enter L1 and click **Send ASCII.** The LED should turn ON.

Suggestion for additional work
You could enter the program name in the following format inside file **/etc/rc.local** so that the program starts automatically every time the Raspberry Pi 5 restarts:

python /home/pi/blueled.py &

When you finish your project, don't forget to remove the above line from file **/etc/rc.local**; otherwise the program will run every time your Raspberry Pi 5 is restarted. You should also shut down your Raspberry Pi 5 orderly instead of just removing the power cable. The command to shut down orderly is:

pi@raspberrypi:~ $ **sudo shutdown now**

15.4 Arduino UNO – Raspberry Pi 5 Bluetooth communication

The Arduino Uno has no built-in Bluetooth module. You have to use an external Bluetooth module to communicate with other devices via the Bluetooth protocol. You can, however, easily use a serial Bluetooth module, such as the HC-06. In the next section, you will de-

velop a project and learn how to connect a HC-06 type low-cost Bluetooth module to your Arduino UNO and communicate with the Raspberry Pi 5.

15.4.1 Project 3 - Communicating with an Arduino UNO over Bluetooth

Description: In this project, a button is connected to the Arduino UNO. Also, a +3.3 V relay is connected to the Raspberry Pi 5. Pressing the button on the Arduino sends command **L1** to Raspberry Pi 5 which then activates the relay for 5 seconds. Similarly, sending **L0** deactivated the LED. The aim of this project is to show how the Arduino UNO and Raspberry Pi 5 can communicate by using an external Bluetooth module on the Arduino UNO.

The HC-06 Bluetooth module

HC-06 is a low-cost, popular 4-pin serially controlled module with the following pins (see Figure 15.10):

Figure 15.10 The HC-06 Bluetooth module

HC-06 is a serially controlled module, having the following basic specifications:

- +3.3 V to +6 V operation
- 30 mA unpaired current (10 mA matched current)
- Built-in antenna
- Band: 2.40 GHz – 2.48 GHz
- Power level: +6 dBm
- Default communication: 9600 baud, 8 data bits, no parity, 1 stop bit
- Signal coverage: 30 feet
- Safety feature: Authentication and encryption
- Modulation mode: Gauss frequency shift keying

Block diagram: Figure 15.11 shows the block diagram of the project.

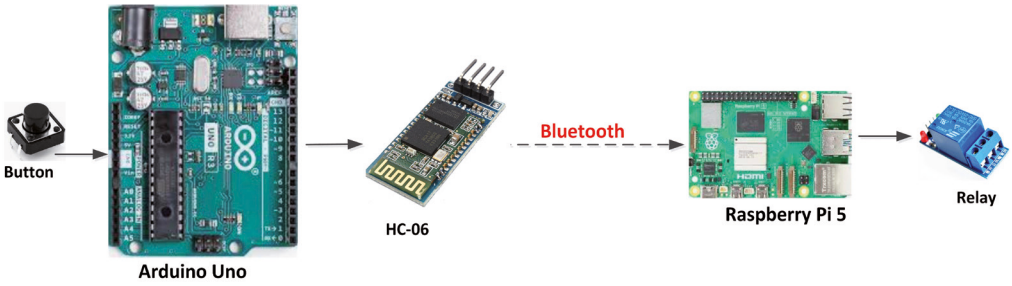

Figure 15.11 Block diagram of the project

Circuit diagram: The circuit diagram is shown in Figure 15.12. The button is connected to pin 2 of Arduino UNO through a pull-up resistor. The relay is connected to port GPIO 21 of the Raspberry Pi 5. HC-06 is a serial device with TX and RX pins. Only the RX input of the HC-06 is used, since in this project you only send data to the Bluetooth module. The output pin voltage of the Arduino UNO is +5 V, but HC-06 is not 5-V compatible. Therefore, a resis-

tive voltage divider circuit is used to lower the Arduino voltage to +3.3 V. TX pin of HC-06 is connected to pin 3 of Arduino UNO (pin 3 is configured as a software serial port in software)

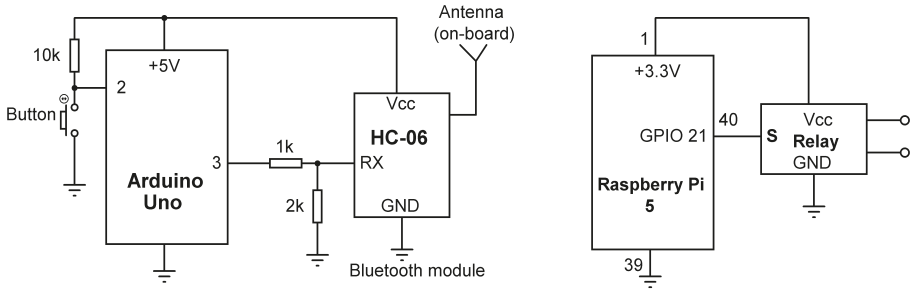

Figure 15.12 Circuit diagram of the project

Connecting to Arduino over Bluetooth

You will find the MAC address of the HC-06 and then use this address to connect to it. The default HC-06 passcode is **1234**. The steps to find the MAC address are:

- Construct the Arduino circuit (Figure 15.12) and apply power so that HC-06 can be accessible. The red LED on HC-06 will be flashing to indicate that it is not currently connected to any device.

- Make your Raspberry Pi 5 Bluetooth discoverable:

  pi@raspberrypi: ~ $ **sudo hciconfig hci0 piscan**

- Start the Bluetooth tool:

  pi@raspberrypi:~ $ **bluetoothctl**

- Turn Bluetooth ON:

  [bluetooth]# **power on**

- Configure Bluetooth to run:

  [bluetooth]# **agent on**
  [bluetooth]# **default-agent**

- Make device discoverable:

  [bluetooth]# **discoverable on**

- Scan for nearby Bluetooth devices, you may have to wait several minutes:

  [bluetooth]# **scan on**

- Enter command **devices** to see the nearby Bluetooth devices. You may have to wait several minutes for the display to update. You should see the HC-06 listed with its MAC address.

  [Bluetooth]# **devices**

  In this example, the author's HC-06 was identified with the MAC address: **98:D3:91:F9:6C:19**

- After finding its MAC address, you may get information about the HC-06 by entering the following command:

  [Bluetooth]# **info 98:D3:91:F9:6C:19**

  Which may be displayed as in Figure 15.13:

```
[bluetooth]# info 98:D3:91:F9:6C:19
Device 98:D3:91:F9:6C:19 (public)
        Name: HC-06
        Alias: HC-06
        Class: 0x00001f00
        Paired: yes
        Trusted: yes
        Blocked: no
        Connected: no
        LegacyPairing: yes
        UUID: Serial Port
        RSSI: -29
```

Figure 15.13 Getting information on HC-06

If **Trusted: no** is displayed, enter command **trust 98:D3:91:F9:6C:19**

Exit the Bluetooth tool by entering **Ctrl+Z** and then enter the following statement to make a connection in command mode (enter your own HC-06 MAC address):

- pi@raspberrypi:~ $ **sudo rfcomm connect hcio 98:D3:91:F9:6C:19 &**
- Enter **Ctrl+C** to exit

You should be connected now and the red LED on HC-06 should stop flashing. You are now ready to develop your programs for the Arduino UNO and Raspberry Pi 5.

Raspberry Pi 5 program: Figure 15.14 shows the Raspberry Pi 5 program (**zeroprog.py**). In this program, we will not be using the Bluetooth library. When we connect to the HC-06, a virtual serial terminal named **/dev/rfcomm0** is created on the Raspberry Pi 5. You can read the commands sent by the HC-06 by opening this serial port and then reading the data.

Serial port **/dev/rfcomm0** is opened with the baud rate set to 9600 (default baud rate of HC-06 is 9600) using the following statement:

ser = serial.Serial(port='/dev/rfcomm0', baudrate=9600)

Data sent by HC-06 is then read using statement:

data = ser.read()

If the received data is b'1' then the relay is activated for 5 seconds by the following statements:

```
RELAY.on()
sleep(5)
RELAY.off()
```

```
#=======================================================
#              RELAY CONTROL BY BLUETOOTH
#              ==========================
#
# In this project a Relay is connected to GPIO 21.The
# Relay is controlled by sending command from an Arduino
# Uno using a Bluetooth apps.
#
# Valid command is: «1»
#
# Author: Dogan Ibrahim
# File  : zeroprog.py
# Date  : October, 2023
#=======================================================
import serial
from time import sleep
from gpiozero import LED

#
# Relay is on GPIO 21, configure as output and turn OFF
#
RELAY = LED(21)                         # Relay at port 21
RELAY.off()                             # Relay off

#
# Attach to virtual serial port /dev/rfcomm0
#
ser = serial.Serial(port='/dev/rfcomm0', baudrate=9600)

#
```

```
# Now receive commands and decode
#
try:

   while True:
     data = ser.read()                # receive comman
     if data == b'1':                 # 1?
         RELAY.on()                   # activate Relay
         sleep(5)                     # 5 seconds
         RELAY.off()                  # Relay OFF

except KeyboardInterrupt:             # Interrupt
     RELAY.off()                      # deactivate Relay
```

Figure 15.14 Program: zeroprog.py

Arduino UNO program: Figure 15.15 shows the Arduino Uno program (**ardprog**). A software serial port is used in this program, where pin 3 is configured as the TX pin and pin 4 as the RX pin (RX is not used in this project). **Button** is then assigned to port 2. Inside the **setup()** function, serial port baud rate is set to 9600 and **Button** is configured as an input pin. Inside the main program loop, the program waits until the button is pressed and then released. '**1**' is then sent to the HC-06, where Raspberry Pi 5 will turn ON the relay when it receives this command.

```
/*================================================
      ARDUINO UNO - RASPBERRY PI 5
      ============================

In this project a button is connectd to Arduino Uno
pin 2. HC-06 Bluetooth module is connected to pin 3
The program sends command "1" over the Bluetooth when
the button is pressed

Autjor: Dogan Ibrahim
File  : ardprog
Date  : October, 2023
================================================*/
#include <SoftwareSerial.h>
SoftwareSerial MySerial(4, 3);        // rx, tx

int Button = 2;                       // Button at pin 2

void setup()
{
  MySerial.begin(9600);               // Baud rate
  pinMode(Button, INPUT);             // Button is input
```

```
}

//
// MAin program looop
//
void loop()
{
  while (digitalRead(Button) == 1);   // Button not pressed
  while (digitalRead(Button) == 0);   // Button not released
  MySerial.print("1");                // Send "1"
  delay(1000);
}
```

Figure 15.15 Program: ardprog

Testing

- Run the Raspberry Pi 5 program:

  pi@raspberrypi:~ $ **python zeroprog,py**

- Compile and upload the Arduino UNO program

- Press and release the button. The relay should turn ON for 5 seconds

15.4.2 Project 4 – Play audio (e.g. music) on Bluetooth speaker via Raspberry Pi 5

Description: In this project, we will play music on an external Bluetooth speaker via our Raspberry Pi 5. We will store our MP3 music files on the Raspberry Pi 5 and then play them on the Bluetooth speaker.

Before you can send audio to a Bluetooth speaker, you have to have a program on the Raspberry Pi 5 that can play audio files (e.g. MP3 files). In this project, we will be using the popular **VLC Media Player** program on the Raspberry Pi 5. The steps to install VLC are:

- pi@reaspberrypi:~ $ **sudo apt-get update**
- pi@reaspberrypi:~ $ **sudo apt-get upgrade**
- pi@raspberrypi:~ $ **sudo apt-get install vlc**

- Wait until the VLC program is installed. You need to have MP3 files to test the project. Download or copy some of your favourite MP3 music files to your Raspberry Pi 5 (e.g. to the default home directory **/home/pi** or to a newly created directory).

You now have to pair with the Bluetooth speaker and connect to it. We will do this from the Desktop. The steps are:

- Start **Desktop** on your Raspberry Pi 5
- Click the **Bluetooth icon** at the top right-hand corner of Desktop and select to turn ON Bluetooth
- Click on the **Bluetooth icon** and set to make it discoverable
- Click on the **Bluetooth icon** and click **Add Device** to pair and add your speaker, or if your speaker is already listed then click on it and click to **Connect** (in the author's example, the Bluetooth speaker had the name **BT-888**). See Figure 15.15.

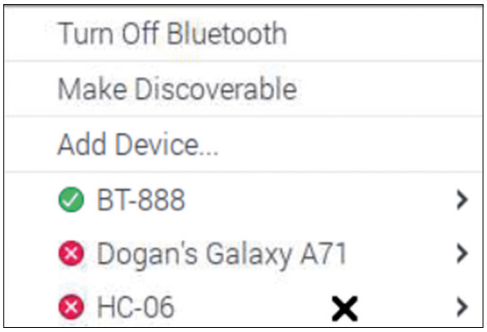

Figure 15.16 Click on Bluetooth speaker device to connect to it

- We are now connected to the speaker. The next thing to do is to direct our audio output to the speaker. Right-click on the **Volume icon** at the top right-hand corner of the **Desktop** and select your Bluetooth speaker name (e.g. **BT-888** in the author's case).
- Open **File Manager** in Desktop (**Accessories** → **File Manager**) and double-click on your MP3 file. The Bluetooth speaker should start playing your chosen music.
- Click **Media** → **Quit** to stop playing the music and exit from VLC.
- Click **File** → **Close Window** to exit **File Manager**

Suggestion: You can create a Play List using the VLC Media Player and store your favourite music files in this list. You can then play the list.

Chapter 16 • Raspberry Pi 5 Camera Projects

16.1 Overview

In this chapter, you will be developing various camera projects using the Raspberry Pi 5. The early sections of the chapter describe how to install and use the camera in still picture mode. In later sections, you will develop more interesting camera-based projects using the Python programming language.

There are several Raspberry Pi camera modules. Version 1 was released in 2013, and it was a 5-megapixel model, which is not available anymore from Raspberry Pi. This was followed by Version 2, which was 8-megapixels and was released in 2016. The latest model Version 3 has 12 megapixels and was released in 2023. Additionally, 12-megapixel high-quality cameras for use with external lenses with CS or M12 type mounting were released in 2020 and 2023 respectively. Raspberry Pi cameras are available as either visible-light-type or infrared-type for night vision. All cameras can take high-resolution pictures along with HD 1080p video, and they can all be controlled by software. See the following link for further information on Raspberry Pi cameras:

https://www.raspberrypi.com/documentation/accessories/camera.html#about-the-camera-modules

Note: The Raspberry Pi 5 camera socket is a 15-pin socket, where most of the Raspberry Pi cameras have 22 pins. It may therefore be necessary to purchase a 15-pin to 22-pin cable to connect your camera to the Raspberry Pi 5 (see Figure 16.1).

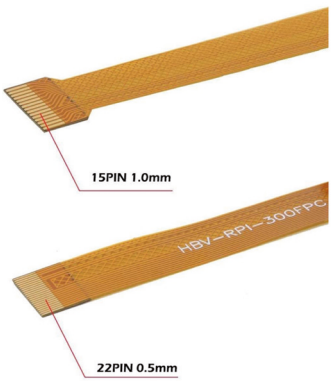

Figure 16.1 15-pin to 22-pin camera cable

16.2 Installing the Camera

The steps to install the camera on your Raspberry Pi 5 are as follows:

- Make sure that your Raspberry Pi 5 is switched off
- Locate one of the camera ports (Figure 1.1)

- Pull up the plastic cable holder slider of the camera port gently by holding from both ends. The cable holder slider will open
- Insert the 15-pin camera ribbon cable (with the connector side facing the white side of the connector) into the cable holder
- Push down the plastic cable holder slider so that it locks and holds the cable
- Connect the 22-pin side of the cable to your camera
- Apply power to your Raspberry Pi 5

You should now be able to test your camera interface and carry out camera operations to capture still images and video frames. Notice that you must be in Desktop GUI mode with a monitor directly attached to Raspberry Pi 5 through the micro-HDMI cable to use the camera. *The author had problems using the camera remotely using his PC even though he was in Desktop GUI mode*. Also, the old camera commands **raspistill**, r**aspiyuv**, **raspivid**, and **raspividyuv** are now obsolete and cannot be used with the Raspberry Pi 5.

In this chapter, a Version 2 visible-light camera is used by the author.

16.3 Project 1 - Still camera commands

In this project, you will be investigating and using the various camera commands to capture still images.

16.3.1 libcamera

libcamera is the new camera software which supports the four Raspberry Pi cameras: V1: OV5647, V2: IMX219, V3: IMX708, and third-party sensors such as IMX290, IMX327, OV9281, IMX378. The following libcamera commands are supported:

libcamera-hello: this command starts a camera preview and displays it on the screen for 5 seconds

libcamera-jpeg: this command captures high resolution JPEG still images

libcamera-still: this is a more complex still camera command which emulates features of the original old raspistill command

libcamera-vid: this is a video capture command

libcamera-raw: this command captures raw (unprocessed) frames directly from the camera sensor

libcamera-hello: This command starts the camera and displays a preview window for 5 seconds. The –t <duration> option enables the user to select for how long the window should be displayed, where <duration> is in milliseconds. For example, the following acti-

vates the camera for 10 seconds:

```
libcamera-hello –t 10000
```

To activate the camera indefinitely, use the command:

```
libcamera-hello –t 0
```

Use Ctrl+C to stop the camera

The following options are available:

Options
The following website displays all the options:

https://www.raspberrypi.com/documentation/computers/camera_software.html#common-command-line-options

The options are in 3 groups: common options, still camera options, and video options. Some options are:

Common options:

--help	display help information
--version	display the software version
--list-cameras	list the cameras available for use
--camera	select a camera to use
--config –c <filename>	read camera options from file <filename>
--preview (--p)	preview window settings <x,y,w,h>
--fullscreen (or --f)	fullscreen preview
--n	suppress the preview. This is very useful for quickly taking images without preview
--info-text	set window title bar to <text>
--shutter	set the exposure time in microseconds
--awb	set the AWB mode
--output	specify the output filename to save the image
--o	same as above

Still camera options:

--quality	JPEG quality number (93 default, 100 maximum)
--exif	add extra EXIT tags
--timelapse	time interval between time-lapse captures in milliseconds
--datetime	use date format for the output filename
--timestamp	use system timestamp for the output filename
--keypress (or --k)	capture image when the Enter key is pressed

--encoding (or --e)	set the image encoding (jpg, png, bmp, rgb, yuv420)
--raw (or --r)	save raw file

Video options:

--quality (or --q)	JPEG quality
--bitrate (or --b)	H.264 bitrate
--codec	encoder to use (h264, mjpeg, yuv420, libav)
--keypress (--k)	toggle between recording and pausing
--split	split multiple recordings into separate files

Some examples are given below:

libcamera-hello --list

Figure 16.2 shows the available cameras.

```
pi@raspberrypi:~ $ libcamera-hello --list
Available cameras
-----------------
0 : imx219 [3280x2464] (/base/axi/pcie@120000/rp1/i2c@80000/imx219@10)
    Modes: 'SRGGB10_CSI2P' : 640x480 [206.65 fps - (1000, 752)/1280x960 crop]
                             1640x1232 [41.85 fps - (0, 0)/3280x2464 crop]
                             1920x1080 [47.57 fps - (680, 692)/1920x1080 crop]
                             3280x2464 [21.19 fps - (0, 0)/3280x2464 crop]
           'SRGGB8' : 640x480 [206.65 fps - (1000, 752)/1280x960 crop]
                      1640x1232 [41.85 fps - (0, 0)/3280x2464 crop]
                      1920x1080 [47.57 fps - (680, 692)/1920x1080 crop]
                      3280x2464 [21.19 fps - (0, 0)/3280x2464 crop]

pi@raspberrypi:~ $
```

Figure 16.2 Available cameras

libcamera-jpeg –o mycamera.jpg image is captured and saved in file **mycamera.jpg**. To view the captured image, click to open **File Manager** in Desktop GUI mode, scroll down and right-click on the image file **mycamera.jpg** and select **Image Viewer**. You should see the image displayed on your monitor. An example display is shown in Figure 16.3

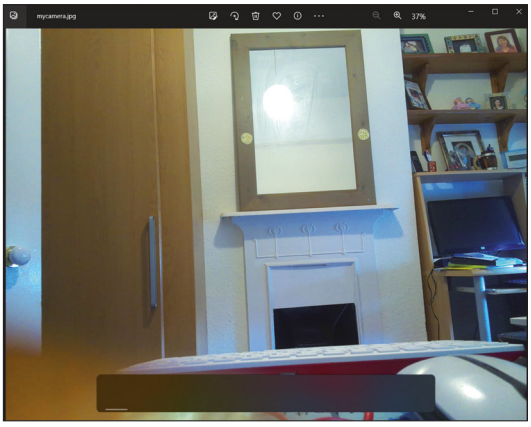

Figure 16.3 example display on the monitor

Notice that there are several menu options when the image is displayed. Selecting the **Edit** menu option, you can copy, flip horizontal, flip vertical, rotate clockwise, rotate anti-clockwise, etc. Selecting the **View** option, you can see in full screen, do a slideshow, zoom in, zoom out, etc. Selecting the **Go** option, you can see the previous or the next image, or the last image etc.

libcamera-jpeg –o test.jpg –t 5000 --width 640 --height 480 will display the preview for 5 seconds, save the image in file **test.jpg** where the image will be in VGA format

libcamera-jpeg –o test.jpg –t 3000 --shutter 20000 --gain 1.5 will display the preview for 3 seconds, set the shutter speed to 20 ms and gain to 1.5x

libcamera-still is similar to libcamera-jpeg, but it supports more of the legacy raspistill options. Some examples are:

libcamera-still –o test.jpg image is captured and saved in file **test.jpg**.

libcamera-still –e bmp –o test.bmp saves the file as BMP format in file **test.bmp**

libcamera-still –r –o test.jpg saves in raw format

16.4 Project 2 – Building a time-lapse camera – Who is in my parking place?

Description: In this project, we have positioned our camera to take pictures of our parking place. We will be taking still images every minute (60,000 ms) for the duration of ten minutes (600,000 ms).

The stages in creating a time-lapse camera session are as follows:

- Fix the camera at the place of interest
- Take a picture of the object to make sure that it is in the field of view and is in focus
- Take pictures of the object at regular intervals
- Combine the pictures into a movie file
- Play the movie file

Required Commands: In time-lapse camera sessions, you usually have to store large number of pictures taken at regular intervals. Before doing this, you have to know the size of a picture, how often the pictures will be taken, and the space available on your storage device (e.g. the Raspberry Pi SD card). The author has taken a number of 3280 × 2434 (8 megapixel) pictures of the same environment at different quality levels using the following command:

```
libcamera-still  -t 600000 -o Mypics%d.jpg –timelapse 60000
```

The file sizes are as follows:

Quality level (-q setting)	Exact File size (bytes)	Approximate file size (MB)
100	4,799,812	4.8
50	4,149,210	4.2
25	2,484,031	2.5
10	739,408	0.74
5	281,971	0.3

It was observed by the author that there was not much noticeable change in the visible quality of the images at different quality levels. It is therefore recommended to take the picture at say 10% quality level (option **–q 10**) where the size of a file is around 0.74 MB. If **f** is how often we want to capture the images in minutes, and **d** is the total duration of the capturing process in minutes, then the required storage is, **s = 0.74d/f MB**. During this period, **n = d/f** images will be taken and also **n** files will be created. As an example, if we want to capture images every minute for the duration of 10 minutes, then the required storage will be s = 0.74 × 10/1 = 7.4 MB and 9 images will be taken with 9 files created.

The required commands to start the time-lapse in this project are as follows. Images will be taken every minute for the duration of 10 minutes. A directory called **mypics** is created and all the created image files are stored in this directory with the filenames starting with **Mypics** and including a four-digit ascending number:

```
pi@raspberrypi:~ $ mkdir mypics
pi@raspberrypi:~ $ cd mypics
pi@raspberrypi:~/mypics $ libcamera-still  -t 600000 -o Mypics%04d.jpg
--timelapse 60000 –q 10
```

Figure 16.4 shows the files created in directory **mypics**. The image files are created with the names such as **Mypics0000.jpg**, **Mypics0001.jpg**, **Mypics0002.jpg**, … etc.

```
pi@raspberrypi:~/mypics $ ls
Mypics0000.jpg  Mypics0002.jpg  Mypics0004.jpg  Mypics0006.jpg  Mypics0008.jpg
Mypics0001.jpg  Mypics0003.jpg  Mypics0005.jpg  Mypicd0007.jpg
pi@raspberrypi:~/mypics $ ls
```

Figure 16.4 Created image files

After all the still JPEG files have been created, you may want to join the files together and create a video file. This can be done in several ways. In this section, we shall be using the software called **ffmpeg** which is already installed on Raspberry Pi 5.

Now we can join our JPEG files together into a video file. Enter the following commands to move to the directory where the files are, and then join the still image files to create the video file **Mytimelapse.mp4**:

```
pi@raspberrypi:~ $ cd mypics

pi@raspberrypi:~/mypics $ ffmpeg -framerate 1 –i Mypics%04d.jpg –c:v
libx264 –r 30 Mytimelapse.mp4
```

The above command takes all the input images, **-i Mypics%04d.jpg**. This will search for the image with the lowest digit and sets that as the starting image. It will then increment that number by one and if the image exists, it will be added to the sequence. Option **-framerate 1** is used to define how fast the pictures are read in, in this case, one picture per second. Omitting the frame rate will default to 25. **-r 30** is the frame rate of the output video. Again, if it defaults to 25 if not defined. The **-c:v libx264** specifies the codec to use to encode the video. **x264** is a library used for encoding video streams into the **H.264/ MPEG-4 AVC** compression format.

You can play the output video (**Mytimelapse.mp4**) file by double-clicking on it and selecting the **VLC media player** in your Desktop GUI. You can, if you wish, copy the created video file to your PC after installing and using the file copy software called **winSCP** on your PC. After the file is copied to your PC, you can play it using various video player software:

This completes the design of our time-lapse camera processing.

Scheduling the Time-lapse

The process we have described to capture still time-lapse images requires the Raspberry Pi 5 to be connected to a computer (or a monitor) so that the commands can be issued from the command line. In this section, you will see how to schedule the time-lapse process so that it starts automatically after the Raspberry Pi 5 is powered up, without the need to connect a monitor or a computer. This can be done using the software tool called **crontab** as described below.

With the command **crontab** you can specify a list of tasks to be scheduled at specified times of the day, and at specified days of the week. You should run crontab by entering the following command:

```
pi@raspberrypi:~ $ crontab –e
```

The first time you run **crontab** you should be asked to select an editor. Select **nano** (option 1) as this is the easiest editor to use. **crontab** consists of six components for minutes, hours, day of the month, month of the year, day of the week, and the command to be executed, organized as shown in Figure 16.5.

```
# m h  dom mon dow   command
# * * * * *  command to execute
# ┬ ┬ ┬ ┬ ┬
# │ │ │ │ │
# │ │ │ │ │
# │ │ │ │ └─────── day of week (0 - 7) (0 to 6 are Sunday to Saturday, or use
names; 7 is Sunday, the same as 0)
# │ │ │ └──────────── month (1 - 12)
# │ │ └─────────────────── day of month (1 - 31)
# │ └────────────────────────── hour (0 - 23)
# └──────────────────────────────── min (0 - 59)
```

Figure 16.5 crontab fields

Some examples are given below (notice that you must give the full path to the script file):

* * * * * sh /home/pi/test.sh	run test.sh every minute
*/2 * * * * sh /home/pi/test.sh	run test.sh every 2 minutes
*/20 * * * * sh /home/pi/test.sh	run test.sh every 20 minutes
0 0 * * * sh /home/pi/test.sh	run test.sh every day at midnight
* * * * 1,3 sh /home/pi/test.sh	run test.sh every minute on Mondays and Wednesdays
* 1 * * 1 sh /home/pi/test.sh	run test.sh every Monday at 1:00 a.m.
30 9 * * 5 sh /home/pi/test.sh	run test.sh every Friday at 9:30
0 6 * * * sh /home/pi/test.sh	run test.sh every day at 6 a.m.

As well as single numbers for each of the first 5 parameters, you can also use the following special formats:

- A sequence of numbers, separated by a comma (e.g. 0,20,40,42)
- A range (4–9)
- A sequence of ranges (e.g. 0–10,30-50)
- An asterisk, meaning 'all' (e.g. *)
- Every n'th time by adding the /c character (e.g. */2 for every 2nd minute)

The **crontab generator utility** helps to generate a **crontab** table for the specified scheduling times. This utility can be accessed from the following website on your PC:

https://crontab-generator.org/

An example use of the crontab generator utility is shown in Figure 16.6 where the utility is used to set the scheduling time to be every 5 minutes. The script file is set to **/home/pi/test.sh**, which should be entered in the field **Command to Execute** at the lower part of the screen. Then, click **Generate Crontab Line**. The required crontab command and sample times that the script will run at are shown at the top of the crontab generator as shown in Figure 16.7.

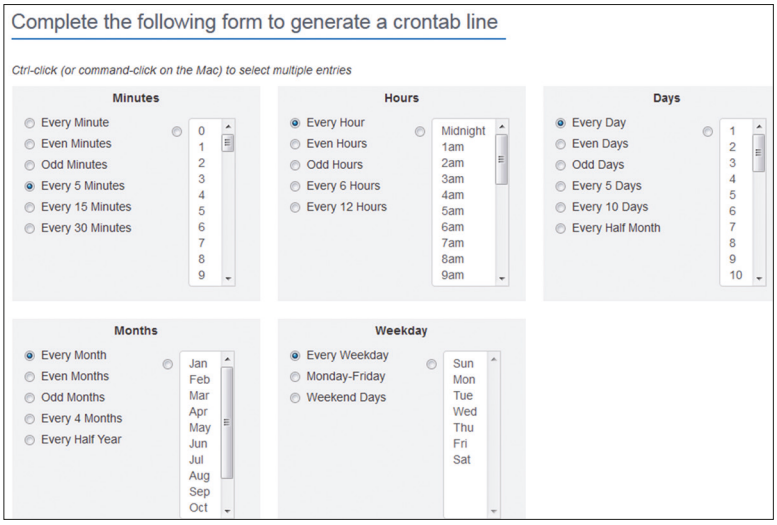

Figure 16.6 crontab generator utility set to one minute past every hour

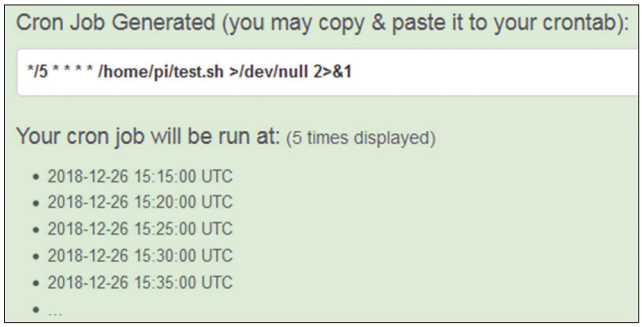

Figure 16.7 The required command and sample scheduling times

Now, going back to our project, we wish to run a script file every minute to capture still images. The script is given the filename **timelapse.sh.** The crontab command for this project should be as follows:

```
*/1 * * * * sh /home/pic/mypics/timelapse.sh 2>&1
```

Notice that **2>&1** at the end of the script ensures that email messages are not sent after the command is executed. Here, we redirect 2 (stderr) to 1 (stdout) and since the output is redirected to a file, it will not generate emails of outputs. Exit from the crontab table by pressing **Ctrl** followed by **Y** and return to save the new file.

The scheduled tasks can be listed with the following command. Enter the command, and you should see your script file scheduled to run:

```
pi@raspberrypi:~ $ crontab –l
```

Next, we have to create the script file **timelapse.sh** in the folder **mypics**. This can be done using the **nano** editor. The steps are:

- Navigate to the folder mypics: pi@raspberrypi:~ $ **cd mypics**
- start the **nano** editor: pi@raspberrypi:~/mypics $ **nano timelapse.sh**
- Enter the following lines into the blank file. Variable DATETIME will extract the current data and time every time (every minute) the script runs. Date and time-stamped images will then be taken and stored in files with extensions **.jpg**. The images will be captured with 10% resolution **(-q 10**):

```
DATETIME=$(date +"%d-%m-%Y_%H%M%S")
libcamera-jpeg –q 10 –o /home/pi/mypics/$DATETIME.jpg
```

- Exit from the **nano** editor by pressing **Ctrl X** followed by **Y** and return to save
- Display the contents of the file to make sure that you have the correct lines in the file:

    pi@raspberrypi:~/mypics $ **cat timelapse.sh**

- Make sure that your script file runs correctly. Enter the command: **sh timelapse.sh**
- and you should see a new JPEG file with the filename containing the date and time.
- You should now reboot your Raspberry Pi 5. The still images will be taken every minute after your Raspberry Pi 5 starts. After taking all the necessary images, don't forget to edit the crontab table and remove the **timelapse.sh** entry. Then reboot your Raspberry Pi 5 (pi@raspberrypi:~/mypics $ **sudo reboot**) so that no more images will be taken.
- Check folder **mypics** to make sure that you have the image files. You can use the following command to list the files in the folder:

    pi@raspberrypi:~/mypics $ **ls**

- After collecting all your images, you may want to join all the image files together and create a video file as described earlier.
- Don't forget to run the crontab –e and remove the scheduling

16.5 Project 3 - Video camera commands

libcamera-vid is the video capture command. By default, it uses the Raspberry Pi's hardware H.264 encoder. It will display a preview window and write the encoded bit stream to the specified output. For example, to write a 10-second video to file **test.h264**, use the command:

```
libcamera-vid -t 10000 -o test.h264
```

Video is recorded as raw H264 format, which is incompatible with many video players. The resulting file can be played using the VLC media player program.

Note that this is an unpacked video bit stream, it is not wrapped in any kind of container format (such as an MP4 file). The --save-pts option can be used to output frame timestamps so that the bit stream can subsequently be converted into an appropriate format using a tool like mkvmerge.

```
libcamera-vid -o test.h264 --save-pts timestamps.txt
```

and then if you want an mkv file:

```
mkvmerge -o test.mkv --timecodes 0:timestamps.txt test.h264
```

the following command can be used for mpeg output format:

```
libcamera-vid -t 10000 --codec mjpeg -o test.mjpeg
```

16.6 Project 4 – Who is ringing my doorbell?

Description: In this project, the camera is mounted on our front door. When the doorbell button is pressed, the camera automatically takes a picture of the person ringing the doorbell, and this picture is sent to an Android smartphone. Additionally, a relay is turned ON for 5 seconds to activate a doorbell.

Block Diagram: Figure 16.8 shows the block diagram of the project.

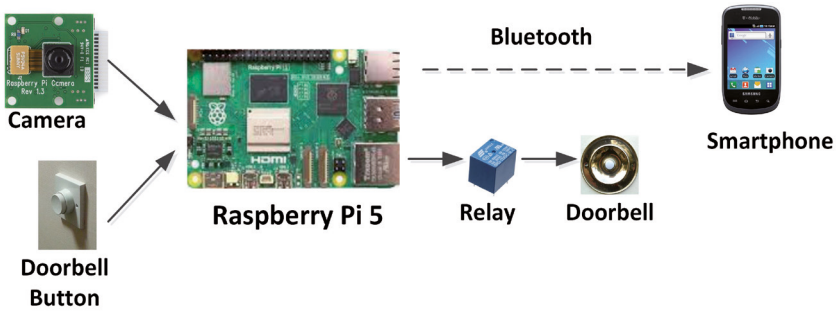

Figure 16.8 Block diagram of the project

Circuit Diagram: The circuit diagram of the project is shown in Figure 16.9. The doorbell button and the relay are connected to port pins GPIO 20 and GPIO 21 respectively.

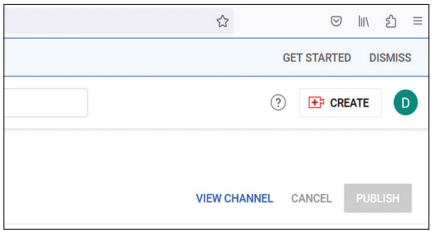

Figure 16.9 Circuit diagram of the project

Program listing: In this program, you will be using the **OBEX Object Push** in Raspberry Pi 5 command mode to send pictures to your smartphone using Bluetooth. Before using OBEX Object Push, you have to find out the MAC address and the channel number of your smartphone. The steps on an Android phone are given below:

- Enable Bluetooth on your smartphone
- Go to **Settings**, then click **System**
- Click **About phone**, then click **Status**
- You should see the Bluetooth MAC address listed. On the author's phone, the MAC address was **50:50:A4:0F:62:3F**, as shown in Figure 16.10

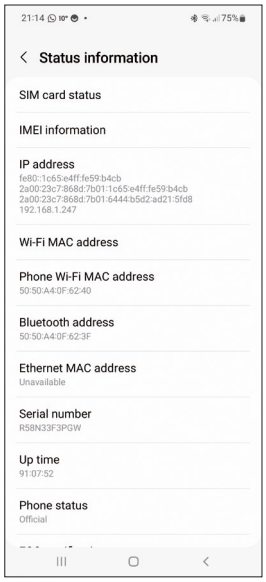

Figure 16.10 Displaying the Bluetooth MAC address

We now have to find the channel number for the Bluetooth communication. Enter the following command on your Raspberry Pi 5 and look for the channel number under section Service Name: OBEX Object Push

pi@raspberrypi:~ $ **sdptool browse 50:50:A4:0F:62:3F**

Figure 16.11 shows the channel number as 12 in this example.

```
Browsing 50:50:A4:0F:62:3F ...
Service Search failed: Invalid argument
Service Name: OBEX Object Push
Service RecHandle: 0x1000b
Service Class ID List:
  "OBEX Object Push" (0x1105)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 12
  "OBEX" (0x0008)
Profile Descriptor List:
  "OBEX Object Push" (0x1105)
    Version: 0x0102
```

Figure 16.11 Read the channel number

Now, we have to install the OBEX software onto our Raspberry Pi 5. Enter the following command:

pi@raspberrypi:~ $ **sudo apt-get install obexftp**

We are now ready to develop our program (**bell.py**), which is shown in Figure 16.12. The button and relay are initialized at the beginning of the program and **os** module is imported since we want to run a shell command from within our Python program. The program then enters an endless loop using the **while** statement. Inside this loop, the program waits until the button is pressed. At this point, the camera takes a picture using a **libcamera** command and stores in a file called **door.jpg**. The relay is also activated for 5 seconds. The picture is then sent to the smartphone using OBEX.

```
#-----------------------------------------------------------------
#
#                       WHO IS AT MY DOOR
#                       =================
#
# In this program a camera, aa pushbutton switch and a relay are all
# connected to Raspberry Pi 5. The camera is positioned outside the
# door so that it can see whos is outside the door.Pressing the button
#  activates the relay for 5 seconds and then takes a picture of the
#  person outside the door and sends it to a smart phone over Bluetooth.
#
# Program: bell.py
# Date   : October, 2023
```

```
# Author : Dogan Ibrahim
#-------------------------------------------------------------------
from gpiozero import LED, Button
from time import sleep                  # import time library
import os

button = Button(20)                     # Button at GPIO 20
relay = LED(21)                         # Relay at GPIO 21
relay.off()

while True:
  sleep(1)
  if button.is_pressed:                 # If button
     relay.on()                         # Relay on for 5 secs
     sleep(5)
     relay.off()
     os.system("obexftp --nopath --uuid none --noconn --bluetooth
50:50:A4:0F:62:3F --channel 12 -p door.jpg")
  else:
     relay.off()
```

Figure 16.12 Program listing

Figure 16.13 shows how the program is run and its output on the screen. You should wait for the file transfer to complete, since it may take some time. A confirmation message is sent to the smartphone before the file transfer takes place. You should click the **ACCEPT** button to receive the picture as shown in Figure 16.14.

```
pi@raspberrypi:~ $ python bell.py
Suppressing FBS.
Connecting..\done
Sending "door.jpg"...\done
Disconnecting..|done
Suppressing FBS.
Connecting..\done
Sending "door.jpg"...|^C
```

Figure 16.13 Running the program

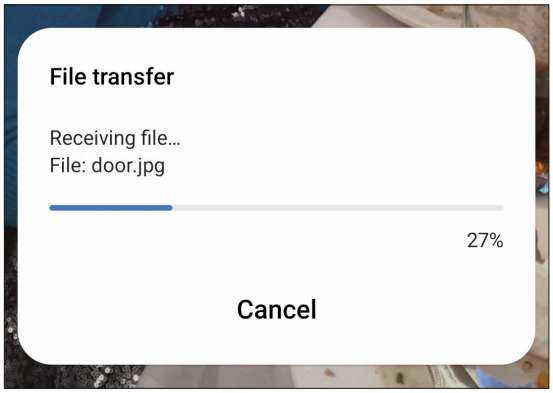

Figure 16.14 Accept the file transfer

Index

W

elektor books

Raspberry Pi 5 Essentials

Program, build, and master over 60 projects with Python

The Raspberry Pi 5 is the latest single-board computer from the Raspberry Pi Foundation. It can be used in many applications, such as in audio and video media centers, as a desktop computer, in industrial controllers, robotics, and in many domestic and commercial applications. In addition to the well-established features found in other Raspberry Pi computers, the Raspberry Pi 5 offers Wi-Fi and Bluetooth (classic and BLE), which makes it a perfect match for IoT as well as in remote and Internet-based control and monitoring applications. It is now possible to develop many real-time projects such as audio digital signal processing, real-time digital filtering, real-time digital control and monitoring, and many other real-time operations using this tiny powerhouse.

The book starts with an introduction to the Raspberry Pi 5 computer and covers the important topics of accessing the computer locally and remotely. Use of the console language commands as well as accessing and using the desktop GUI are described with working examples. The remaining parts of the book cover many Raspberry Pi 5-based hardware projects using components and devices such as

- LEDs and buzzers
- LCDs
- Ultrasonic sensors
- Temperature and atmospheric pressure sensors
- The Sense HAT
- Camera modules

Example projects are given using Wi-Fi and Bluetooth modules to send and receive data from smartphones and PCs, and sending real-time temperature and atmospheric pressure data to the cloud.

All projects given in the book have been fully tested for correct operation. Only basic programming and electronics experience are required to follow the projects. Brief descriptions, block diagrams, detailed circuit diagrams, and full Python program listings are given for all projects described. Readers can find the program listings on the Elektor Store website, www.elektor.com (search for: book title).

Prof Dogan Ibrahim has a BSc (Hons) degree in Electronic Engineering, an MSc degree in Automatic Control Engineering, and a PhD degree in Digital Signal Processing and Microprocessors.

Dogan has worked in many organizations and is a Fellow of the Institution of Engineering and Technology (IET) in UK as well as a Chartered Electrical Engineer. He has authored over 100 technical books and over 200 technical articles on electronics, microprocessors, microcontrollers, and related fields. Dogan is a certified Arduino professional and has many years of experience with numerous types of microprocessors and microcontrollers.

Elektor International Media
www.elektor.com

ISBN 978-3-89576-586-5
9 783895 765865

elektor
design > share > earn